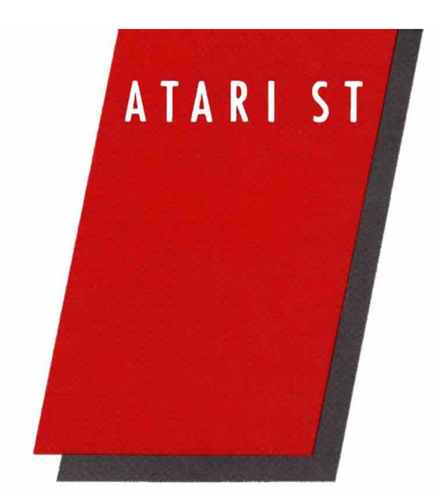




Litzkendorf

DAS GROSSE GFA BASIC 3.0 BUCH

einschließlich aller BASIC-Versionen und Compiler

DATA BECKER

Danksagung

Anfang 2019 begann ich damit, alles an Atari-Hardware zu erwerben, was ich bereits in den 80er/90er-Jahren schon einmal besaß. So auch Bücher, die sich speziell mit dem Thema Programmierung beschäftigen. Ich stellte schnell fest, dass dieses Vorhaben heutzutage schwieriger werden könnte als der Kauf eines Atari ST. Für eine gebrauchte Ausgabe von „Das große GFA BASIC 3.5 Buch“ zahlte ich mehr als den damaligen Neupreis! Mein Wunsch war aber, genau dieses Buch wieder in meiner Sammlung zu sehen. Da das Buch sehr gut erhalten war, mochte ich es auch so belassen und scannte es daher ein. Somit konnte ich nun bequem digital darin blättern, nach Begriffen suchen und Lesezeichen erstellen. Später entstand allerdings der Gedanke, dass diese digitale Version doch eigentlich der Allgemeinheit zur Verfügung gestellt werden sollte. Eine Anfrage zur Genehmigung beim Data Becker Verlag war allerdings nicht mehr möglich, da der Verlag seinen Geschäftsbetrieb zum 31.03.2014 aufgegeben hatte und es auch keine Rechtsnachfolger gibt. Somit machte ich mich auf die Suche nach dem Autor, Uwe Litzkendorf, und fand ihn auch. Mit großer Freude durfte ich feststellen, dass er meine Meinung teilte. Und mehr noch, Uwe machte den Vorschlag, dass ich es nicht nur bei diesem einem Buch belassen sollte, sondern auch weitere seiner geschriebenen Fachbücher zum Thema GFA-Basic digitalisieren dürfte. Ich war begeistert und erklärte mich gerne bereit dieses Projekt durchzuführen. So entstand aus einer einfachen E-Mail Anfrage eine wunderbare Möglichkeit den Inhalt dieser Bücher digital für die Zukunft zu erhalten und allen Interessierten kostenfrei zur Verfügung stellen zu können. Dafür möchte mich bei Uwe noch einmal sehr herzlich bedanken.

Thomas Werner
Bearbeitungsstand: 02. Juni 2020

Neues Vorwort zur Digitalisierung

Es hat lange gedauert! Aber nun hat sich Thomas Werner die viele Arbeit gemacht, den größten Teil meiner Bücher nach, zum Teil über dreißig, Jahren in interaktiver PDF-Form zu digitalisieren und als Public Domain kostenlos online zur Verfügung zu stellen. Nachdem meine Bücher einen langen Dornröschenschlaf hinter sich haben, sollen sie und das immense darin festgeschriebene Wissen jetzt wieder zum nützlichen Leben erweckt werden :O)

Meiner Ansicht nach ist die objektorientierte Programmierung nicht für die Massenanwendung geeignet. Nach meiner Berechnung sind nur ca. 15 Prozent der Bevölkerung in der Lage, mit der OOP sinnvolle Erfolge zu erzielen. Um aber "massenfähig" zu sein, muss eine Programmiersprache mindestens 80 Prozent der Bevölkerung erreichen, damit sich Lehrer und Schüler auch außerhalb von IT-Leistungskursen über die Softwareentwicklung so verständigen können, dass der eine – vermittelbar und auch prüfbar – wenigstens "einigermaßen" versteht, was der andere meint. Andererseits "reißt unweigerlich der Faden" zwischen Ausbildern und Lernenden. Daher ist es auch kein Wunder, dass sich die prozeduralen und damit auch leicht anwendbaren Basic-Programmiersprachen, wie z.B. das sehr populäre GFA-Basic, einer gewissen Renaissance erfreuen.

Diese Tendenz werde ich natürlich als ehemals populärer Bestseller-Autor nach all meinen Möglichkeiten tatkräftig unterstützen. Ich verfüge über mehr als 4000 Seiten umfassenden, ausführlichen und auch unter den wachsamen Augen der Öffentlichkeit profund geprüften Software-Wissens. Dieses Wissen ist auch in der heutigen Zeit absolut nicht überflüssig und veraltet, sondern bildet auch heute noch die Basis für algorithmisches Grundlagenwissen.

Aber damit nicht genug: ich habe zudem beschlossen, eine neue Programmiersprache namens "QS!X©" zu entwickeln. Sie wird sich in vielen Punkten an einfachem Standard-Basic anlehnen. Auf der weltweit überall auf allen Betriebssystemen und in jedem Standardbrowser verfügbaren – und damit 100% cross-kompatiblen – Plattform von HTML5/Canvas wird "QS!X©" als Open Source-Version (ähnlich LINUX) verfügbar sein. Nähere Informationen dazu finden Sie unter:

http://www.litzkendorf.net/invitation_info_d.pdf

Damit ist auch der "Klasse für die Masse"-Philosophie von Frank Ostrowski (dem GFA-Basic-Vater) Rechnung getragen. Wenn denn alles so funktioniert, wie ich es mir vorstelle, wird die weise, sanfte und erzfreundlich geduldige und bescheidene Denkart von Frank Ostrowski auch Jahre nach seinem (viel zu frühen) Ableben noch weltweit merkliche Wirkung tragen! Er bildet dann verdientermaßen – zumindest im IT-Business – die philosophische Grundlage für eine Art "Weltsprache"! Das würde ihm sicher sehr gefallen!

In treuem Gedenken an einen wirklich großen Mann, mit dem ich das Vergnügen hatte, teil- und zeitweise recht eng und vertraut zusammen zu arbeiten und dem mit "QS!X©" auch ein digitales – und verdientes – Denkmal gesetzt wird!

Uwe Litzkendorf
Hildesheim, im Mai 2020

****

Uwe Litzkendorf

Das große GFA-BASIC-Buch

DATA BECKER

6. überarbeitete und erweiterte Auflage 1988

ISBN 3-89011-222-6

DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Text verarbeitet mit Word 4.0, Microsoft

Ausgedruckt mit Hewlett Packard LaserJet II

Druck und Verarbeitung Graf & Pflügge, Düsseldorf

Vorwort

Die Autoren anderer GFA-BASIC-Bücher können es sicher bestätigen: Es ist sehr, sehr schwer, mit einem Programmier-Virtuosen der Klasse eines Frank Ostrowski (der Vater des GFA-BASIC) Schritt zu halten.

In der Entwicklungszeit zu diesem Buch hat mich ab und zu das Bedürfnis befallen, in meinen Computer zu beißen, dem lieben Frank einen Beschwerdebrief bezüglich seiner Arbeitswut zu schreiben und das GFA-BASIC einfach GFA-BASIC sein zu lassen.

Es ist einfach eine so gewaltige Menge an Befehlen und damit verbundenen Gags und Tricks, daß mir manchmal ganz schwindelig wurde. Hauptsächlich in der Endphase, wenn neben der Arbeit auch noch hinzukommt, daß ich wochenlang (ja - wochenlang) nur mit 4 bis 6 Stunden Schlaf pro Tag auskommen mußte. Der Markt drückt, der Verlag drückt, die Konkurrenz drückt, die Händler drücken, das kleiner werdende Bank-Konto drückt, der eigene Qualitäts-Anspruch drückt und die liebe "family" drückt auch noch - also Druck von allen Seiten.

Und dann der Jubelschrei: "Ich hab's geschafft!"

Neben der Möglichkeit, durch diesen "Job" Geld zu verdienen (völlig legitimes Bedürfnis, denn: "money makes the world go round") spielt es für mich eine sehr große Rolle, daß es ein gutes Buch wird (in der Hoffnung auf Ruhm und Ehre, natürlich). Als Maßstab habe ich mir immer vor Augen gehalten, was ich selbst von einem Buch erwarte, um es als gut bezeichnen zu können. Das wichtigste: es muß "gut in der Hand liegen". Darunter verstehe ich ein gutes Gefühl, wenn man darin herumblättert; den Eindruck von Vollständigkeit, Qualtität, guter Struktur, Inhalt usw.

Nun gut - ich habe mein Bestes getan, um diese Bedingungen zu erfüllen. Daß dieses Buch auch Ihnen "gut in der Hand liegt", bleibt mir nur zu hoffen, aber ich habe ein "gutes Gefühl"!

Bei dieser Gelegenheit möchte ich die enormen Geduldsleistungen meiner Frau Liane und - vor allem - meiner 3jährigen Tochter Svenja hervorheben. Ich werde in nächster Zeit viel damit zu tun haben, ihre Entbehrungen nach besten Kräften wieder auszugleichen.

Sie finden in diesem Buch eine umfassende Beschreibung aller BASIC-Befehle und -Funktionen (V2.xx/V3.0), eine Vielzahl kleiner Programme und Prozeduren sowie nützliche Tabellen und Grafiken. Da ich der Meinung bin, daß man GFA-BASIC den Status eines eigenständigen Entwicklungs-Systems nicht mehr streitig machen kann, habe ich mich entschlossen, auch alle System-Funktionen (BIOS, XBIOS, GEMDOS), sowie ST-interne Adressen (System-Variablen) und andere spezifische Eigenschaften des Atari ST (z.B. Disketten-Aufbau, Interrupts, Exceptions etc.) zu beschreiben. Es liegt in meinem Interesse, daß Sie bei der Entwicklung eines Programms ein weitgehend vollständiges Buch zur Verfügung haben und nicht ständig in vier, fünf oder noch mehr verschiedenen Büchern nachschlagen zu müssen. Ein vollständiges BASIC verdient ein vollständiges Buch - ohne wenn und aber.

Weiterhin enthält diese Auflage einen umfangreichen Einsteigerteil sowie die Beschreibung des GFA-BASIC-Editors und des V2.xx-Compilers. Die im Anhang aufgeführten Listen und Tabellen sollen Ihnen helfen, in kurzer Zeit wichtige Informationen (bzw. deren Seitennummer) zu finden, ohne - lästigerweise - dauernd blättern zu müssen. Bei manchen Tabellen habe ich es - um den Zusammenhang zu erhalten - für besser befunden, diese in die Befehls- bzw. Funktionsbeschreibung einzubinden. In solchen Fällen steht Ihnen ein umfassendes Stichwort- Verzeichnis zur Verfügung, anhand dessen Sie (quasi assoziativ) die entsprechenden Textstellen ausfindig machen können. Der Vollständigkeit halber sei hier noch die Liste aller im Buch beschriebenen Prozeduren erwähnt. Bei den meisten dieser Prozeduren habe ich - so weit wie möglich - streng darauf geachtet, daß sie als BASIC-Befehlserweiterungen allgemein einsetzbar sind und tatsächlich eine einfachere und komfortablere Programmierung ermöglichen.

Bei der Programmierung dieser Prozeduren habe ich fast ausschließlich die GFA-BASIC-V2.0/V2.02-Versionen verwendet, da eine zu große Anpassung an die V3.0-Version für meine Begriffe noch nicht ratsam ist. Der V3.0-Compiler wird erst in ca. einem halben Jahr einsatzbereit sein, und bis die meisten von Ihnen ihn dann zu Haus im Regal (bzw. in der Diskettenbox) haben werden, wird mindestens ein weiteres halbes Jahr vergehen. Solange wollte ich gewährleisten, daß Programme, die diese Prozeduren einsetzen, auch noch mit dem V2.xx-Compiler compilierbar sind. Eine spätere Anpassung und Erweiterung der Prozeduren an den V3.0-Compiler dürfte den meisten von Ihnen dann keine größere Mühe bedeuten. Dennoch sind in einigen Prozeduren schon mögliche Änderungen und Erweiterungen in Richtung V3.0 angedeutet. Die in diesem Buch aufgeführten Prozeduren und

Beispiele (Beispiele erst ab ca. sechs Zeilen), sowie das RAMKART-Programm finden Sie auf der Diskette im Buch, wodurch sie für Sie sofort einsetzbar sind. Hierbei wurde auf eine Programm-Kommentierung fast vollständig verzichtet, um alles auf einer einseitigen Diskette unterbringen zu können.

Ich hoffe, daß dadurch den unterschiedlichen Ansprüchen Genüge getan ist.

Nach aller Mühe, die jedes Lernen neuer Inhalte mit sich bringt, werden Sie sicher nach eingehender Lektüre dieses Buches in der Lage sein, für fast alle programmtechnischen Probleme, die Ihnen begegnen werden, eine kompakte Lösung zu finden.

Mein zweites Werk "Der Data Becker Führer GFA-BASIC" bildet hierzu eine nützliche Erweiterung, die Ihnen eine kompakte Beschreibung aller GFA-BASIC-Befehle (inkl. V3.0) im praktischen Taschenformat bietet.

Ich wünsche Ihnen zu Ihrer Programmierarbeit von Herzen viel Erfolg, Ausdauer und allzeit "gut Klick".

Uwe Litzkendorf *Hildesheim, den 4. August 1988*

Inhaltsverzeichnis

1. Zu diesem Buch

Das Anliegen dieses Buches ist es, die ca. 450 Befehle und Funktionen, die nunmehr vom GFA-V3.0-Interpreter zur Verfügung gestellt werden, nach Schwerpunkten zu ordnen, um so das Auffinden der gesuchten Befehlsbeschreibungen nach problemorientierten Gesichtspunkten zu ermöglichen bzw. zu erleichtern. Um Ihnen als Leser eine einheitliche Darstellung zu bieten, habe ich mich an folgende Konventionen gehalten:

Jede Befehls- bzw. Funktionsbeschreibung beginnt mit einer Kopfzeile, die deutlich sichtbar den Befehlsnamen, seine mögliche Abkürzung und eine (sehr knappe) Kurzbeschreibung enthält.

Daran anschließend finden Sie die Syntax, in welcher der Befehl/die Funktion einzusetzen ist. Auf die Beschreibung des Befehls/der Funktion folgt dann gegebenenfalls ein Beispiel oder ein Hinweis auf Beispiele an anderer Stelle.

Innerhalb des Textes wurden für bestimmte Situationen, Vorgaben und Optionen jeweils einheitliche Markierungen benutzt.

< > Wird bei einer Befehlsbeschreibung auf bestimmte Tasten verwiesen, wird ihr Name zur besseren Kenntlichmachung in spitzen Klammern angegeben (z.B. <Shift>, <A>, <Return> oder <Undo>).

[ ] Bei Befehlen, deren Syntax variabel ist, wird ein optionaler Befehlsteil in eckigen Klammern angegeben. Dies bedeutet, daß die Angabe (z.B. [;,'] oder [,Länge]) nur dann im Befehl angegeben werden muß, wenn die damit verbundene Option genutzt werden soll.

{ } Viele GFA-BASIC-Befehle können als Abkürzung angegeben werden. Der Interpreter erweitert diese dann selbständig auf die richtige Form. Sollte zu einem Befehl eine Kurzschreibweise exisitieren, ist diese in der Titelzeile und in der Quick-Referenz innerhalb von geschweiften Klammern angegeben (z.B. { SYS } oder { RET }). Diese geschweiften Klammern werden auch von einigen BASIC-Befehlen (Speicherzugriffe wie CHAR{}, BYTE{} etc.) verwendet. Die Verwechslungsgefahr mit den hier gemeinten Abkürzungsklammern ist jedoch gering.

... Soll eine Folge von Anweisungen innerhalb von Befehlen verdeutlicht werden, geschieht dies anhand einer Punktlinie (z.B. FOR...NEXT).

<< Bei einigen AES-Funktionen sind Rückgabevariablen anzugeben. Um diese eindeutig erkennbar zu machen, ist den entsprechenden Variablennamen ein Doppel-Linkspfeil vorangestellt. Diese Pfeile gehören nicht zur Syntax und sind beim Funktionsaufruf wegzulassen (siehe Kapitel 16: "AES-Bibliothek").

>> Vereinzelt sind Rückgabe-String-Variablen vorzubereiten. D.h., in der Variablen wird bei Funktionsaufruf eine Vorgabe erwartet (z.B. File-Namen) und die Rückgabe erfolgt dann in derselben Variablen. In diesem Fall ist dem Variablennamen zusätzlich ein Doppel-Rechtspfeil nachgestellt (z.B. <<Rück$>>). Auch diese Pfeile sind zur Verdeutlichung gedacht und gehören nicht zur Funktionssyntax.

C-Text
Bei vielen AES-Zugriffen, bzw. innerhalb der Objektstrukturen werden bei Textangaben Strings mit einem abschließenden Null-Byte (C-Konvention) erwartet. Für diese Art von Strings steht das Synonym "C-Text" (siehe CHAR{}).

Grundsätzlich sind in der Syntax-Zeile alle Befehlsnamen in Großbuchstaben, alle Variablen, Parameter und Strings in normaler Schreibweise dargestellt (z.B. OPENW Handle).

Bei den Parameterangaben wurden weitgehend einheitliche Bezeichnungen verwendet:

Adresse
Objektbaumadresse/sonst. Adressen (Adressen werden grundsätzlich als 32-Bit-Integer angegeben).

Anz
Anzahl

Arg
Funktionsargument

Feld
Beliebige Feldbezeichnung.

Back=/Var=
Rückgabedaten bei Funktionen.

Expr/Expr$
Numerischer/alphanumerischer Ausdruck.

Handle
GEM-Window-Handle/-Applikations-Handle.

Index
Index von Feldelementen.

Kanal
Datei-Identifikator.

Nummer
GFA-Window-Nummer.

Objekt
Nummer (Index) eines AES-Objektes.

Text
Beliebige Zeichenkette.

Var/Var$
Beliebiger Variablenname (nicht mit VAR verwechseln!).

Xpos/Ypos
Bildschirmkoordinaten.

Bei Dateiname, Programmname und Ordner ist davon auszugehen, daß ein evtl. erforderlicher Suchpfad in den Namen einzubinden ist.

Unter dem Begriff Ausdruck (s.o. Expr) wird hier eine beliebige Zusammenstellung von Konstanten, Formeln, Texten, Funktionen und Variablen verstanden, die zusammen ein Ergebnis liefern.

z.B. numerischer Ausdruck:

```
A%=B%+((234^2/4.7)*12.95*C%)^2.1317+@Func(Abc%)
```

z.B. alphanumerischer Ausdruck:

```
A$="Text"+STR$(A%*B%)+SPACE$(10)+@Func$(Abc$)+B$
```

In den Beschreibungen von Funktionen wird nicht explizit angegeben, daß die Ergebnisse **aller** (auch selbstdefinierter) Funktionen auf verschiedene Weise ausgewertet werden können, z.B. Zuweisung:

```
Var%=@Func    ->  Selbstdefinierte Funktion
Var%=FRE(0)   ->  BASIC-Funktion (z.B. FRE())
```

z.B. Ausgabe:

```
PRINT @Func  ->  Selbstdefinierte Funktion
PRINT FRE(0) ->  BASIC-Funktion (z.B. FRE())
```

z.B. Abfrage:

```
IF @Func=X   ->  Selbstdefinierte Funktion
IF FRE(0)=X  ->  BASIC-Funktion (z.B. FRE())
```

z.B. Dummy-Aufruf:

```
VOID @Func   ->  Selbstdefinierte Funktion
VOID FRE(0)  ->  BASIC-Funktion (z.B. FRE())
```

In der Syntaxzeile von Funktionen wird in diesem Buch zur Verdeutlichung die Zuweisungsvariante (Var=Funktion()) verwendet. Funktionsaufrufe stehen immer stellvertretend für einen Wert oder String, den diese Funktion liefert. Sie können deshalb wie jeder beliebige Wert oder String verwendet und eingesetzt werden. Alle Funktionen sind im Anhang unter "Alphabetische Befehlsliste" mit einem vorangestelltem (f) gekennzeichnet.

Bei allen Dateizugriffen (außer OUT und INP), die die Angabe einer Kanal-Nummer erwarten, ist die Angabe des Nummernzeichens # optional.

In Version V3.0 kann bei allen ON...GOSUB Name-Befehlen der Teil GOSUB vernachlässigt werden. Er wird vom Interpreter selbständig hinzugefügt (z.B. wird aus ON BREAK Name dann ON BREAK GOSUB Name).

1.1 Zur Diskette im Buch

Um auch ST-Besitzern mit einseitigem Laufwerk alle Programme in diesem Buch (mit mehr als ca. 6 Zeilen) zur Verfügung stellen zu können, wurde es notwendig, die Programme für die Diskette möglichst kurz zu halten. Aus diesem Grunde wurden zunächst einmal alle Kommentare aus den Quelltexten entfernt. Weiterhin sind alle Programme als .LST-File abgespeichert. Gehen Sie deshalb bitte so vor:

Zunächst starten Sie den GFA-Interpreter durch Doppelklick. Nachdem die Menüleiste erscheint, wählen Sie den Punkt MERGE an. Achten Sie bitte darauf, daß ein evtl. im BASIC-Editor befindliches Programm vorher durch New zu löschen ist (evtl. vorher abspeichern). Es erscheint eine Fileselect-Box, in der Sie das gewünschte Programm anklicken. Beachten Sie die Hinweise in den einzelnen Kapiteln, da teilweise noch kleinere Programmteile eingefügt werden müssen. Dazu bewegen Sie den Cursor auf die entsprechende Programmstelle und laden die Teile ebenfalls mit Merge dazu. Diese nun fertiggestellten Programme können Sie sich auf eine eigene formatierte Diskette speichern, um diese Arbeit nur einmal durchführen zu müssen.

Ein wichtiger Hinweis:

Das Beispiel in Kapitel 21.5.2 "Disketten-Formatierung" formatiert automatisch die im Laufwerk befindliche Diskette. Legen Sie daher vor(!) Programmstart eine Leerdiskette ein.

Sie sollten sich auf jeden Fall eine Sicherheitskopie Ihrer Programmdiskette anfertigen und deren Schreibschutz arretieren.

2. Der Atari ST

Der MC 68000 von Motorola, der Hauptprozessor der meisten neuen "16-Bittern", wie z.B. Apple Macintosh, Amiga, QL, aber eben auch in unseres Atari ST, bietet im Vergleich zu den früheren 8-Bit-Prozessoren, wie z.B. dem 6502 im C64, eine Vielzahl an Maschinenbefehlen, die erst durch die 16-Bit-Datenbreite verwirklicht werden konnten und auch gleichzeitig die Fähigkeiten und Geschwindigkeit eines 8-Megahertz-Takters voll zur Geltung brachten.

Was heißt 8 Megahertz? Hertz ist eine aus der Physik bekannte Einheit für Schwingungen pro Sekunde. Unsere Fernseher z.B. arbeiten mit einer Bildwiederholungsfrequenz von 50 Hertz. D.h. jede Bildschirmzeile wird innerhalb einer Sekunde 50mal neu aufgebaut. Für unseren Computer bedeutet das, daß ein spezielles Schwing-Quarz mit einer Frequenz von 8 Millionen Hertz (8 Millionen Mal in der Sekunde!) schwingt, und bei jeder Schwingung ein Schalter-Zustand bearbeitet werden kann. In Kombination mit 16 Daten- und 24 Adreßleitungen ergibt sich daraus eine kaum noch zu erfassende Variabilität.

Man stelle sich eine riesige Lagerhalle vor, in welcher Regale mit insgesamt ca. 16 Millionen (!) Schubladen untergebracht sind. In jeder dieser Schubladen läge eine Information, die bestimmte Auskünfte über die Arbeitssituation im Betrieb gibt. Nun soll jemand innerhalb kürzester Zeit erfassen, welche Information in welcher Schublade liegt und welche Auswirkungen der Inhalt dieser Schublade im Zusammenspiel mit vielen verschiedenen anderen Schubladeninhalten auf die Organisation des Gesamtbetriebes hat.

Das ist, wenn man unser Gehirn als Vergleich nicht in Betracht zieht, unzweifelhaft mit menschlicher Kraft nicht machbar - ein Atari ST kann das. Da er jedoch auch analysieren, rechnen und einordnen muß, würde dies länger als eine Sekunde dauern. Trotzdem reicht seine Geschwindigkeit aus, um z.B. mit dem GFA-Interpreter in weniger als einer 20tel Sekunde eine FOR-NEXT-Schleife mit 1000 Schritten zu durchlaufen. Innerhalb eines Schrittes dieser Schleife muß er intern hunderte von Einzelschritten abarbeiten, die Richtigkeit des Programms in seiner Grammatik überprüfen und die verwendeten Befehle analysieren und zuordnen (interpretieren). Wahrhaft eine gewaltige Leistung. War ein 8-Bitter in seinen Kombinationsmöglichkeiten noch einigermaßen überschaubar, ist ein 16-Bitter real kaum noch zu begreifen.

Glücklicherweise wächst die Fähigkeit mit den Möglichkeiten, die sich dem Interessierten bieten. Einer dieser Interessierten war Frank Ostrowski und kaum ein anderer hat seine Fähigkeiten mit den ihm hier gegebenen Möglichkeiten wachsen lassen.

3. Das GFA-BASIC

Die meisten ST-Besitzer bekamen erst einen Begriff von der Qualität ihres Computers, als Frank Ostrowski vor ca. zwei Jahren sein erstes GFA-BASIC vorstellte.

Es gibt wohl kaum andere Programmiersprachen, mit denen auf so atemberaubend einfache Weise selbst schwierige Probleme lösbar sind wie in GFA-BASIC. Überzeugen Sie sich selbst!

Wir haben nun einen Interpreter, der das Angebot eines MC 68000 in einer für viele nutzbaren Sprache zur Verfügung stellt. Er tritt damit in eine ernstzunehmende Konkurrenz mit der bis heute favorisierten Compilersprache C. Zugegeben, in manchen Beziehungen werden C- und Assemblerprogrammierung Vorrang behalten. Aber die Gruppe derer, die bereit sind, sich mit der komplizierten Compiler- und Assemblertechnik auseinanderzusetzen, wird sich mehr und mehr in Grenzen halten, da dieses BASIC (vor allem die V3.0-Version) den meisten Anforderungen mit Sicherheit Genüge tut.

Wie bei jeder Sprache, muß man auch hier erst einmal das ABC lernen, um fließend sprechen zu können. D.h., man muß die Grundstrukturen, an denen sich die Sprache orientiert, beherrschen, um vom Empfänger (in diesem Fall dem Interpreter) richtig verstanden zu werden. Glücklicherweise haben wir es hier mit BASIC zu tun, dem ja der Ruf anhängt, ein Tausendsassa zu sein, was seine syntaktische Toleranz angeht.

Das GFA-BASIC zwingt - glücklicherweise - zu einer strukturierten Programmierung. Wer sich z.B. mit den Sprachen C, Modula oder Pascal beschäftigt hat, dem werden die Eigenarten der strukturierten Programmierung nichts Neues sein.

In GFA-BASIC wird in jeder Zeile jeweils nur ein Befehl akzeptiert. Außerdem werden die Zeilen vom Interpreter selbsttätig in die entsprechende optische Struktur eingeordnet. Es ist eine wahre Freude, zu sehen, wie sauber und ordentlich ein derart durchstrukturiertes Programm hinterher aussieht. Ein weiterer wichtiger Aspekt ist aber, daß bei der Fehlersuche dadurch eine immens große Zeitersparnis eintritt. Befehlszeilen sind ohne Zeichen-Scrolling auf einen Blick erfaßbar. Zusätzlich wird durch das Einrücken der Zeilen sofort erkennbar, z.B. welches ENDIF zu welchem IF, oder welches NEXT zu welchem FOR gehört.

Der vielleicht wichtigste Vorteil der Struktur-Programmierung ist aber der, daß beiläufig während der Programmerstellung immer wieder kleine Unteroutinen abfallen, die in sich geschlossen sind und dadurch die Möglichkeit bieten, nach und nach eine umfangreiche Bibliothek an Hilfsprogrammen und allgemein verwendbaren Prozeduren zusammenzustellen. Das wäre prinzipiell in anderen Programmierarten genauso möglich, nur ergibt sich hier die Gelegenheit dazu erheblich seltener. Komfortabel wird es dann noch, wenn man diesen Prozeduren (wie in GFA-BASIC) eine fast beliebig lange Parameterliste übergeben kann. Effektiver geht es fast nicht mehr.

Die V3.0-Version hat gegenüber den früheren Versionen erhebliche Veränderungen erfahren. Das beginnt bei der Variablen-Organisation (Einführung von Byte- und Word-Variablen), geht über einen phantastischen Programm-Editor und endet nach vielen weiteren Änderungen bei wesentlich strafferen Strukturierungsmöglichkeiten (SELECT-CASE, ELSE IF, FUNCTION etc.).

Aus diesen neuen Möglichkeiten ergibt sich eine erhebliche Einsparung an Programmtext und zudem eine ebenso erhebliche Steigerung der Geschwindigkeit im Programmlauf sowie bei der Programm-Entwicklung. Einige neue Befehle ermöglichen eine derart einfache Programmierung auch komplizierter Vorgänge (z.B. RC_INTERSECT, RC_COPY), daß es fast zu einem Kinderspiel wird.

Wahrscheinlich das erste, was Ihnen am neuen GFA-BASIC auffallen wird, ist die rasante Geschwindigkeit des Editors beim Suchen, Ersetzen, Blättern und Scrollen. Diesen Super-Editor kann man ohne weiteres schon fast als komplette Textverarbeitung bezeichnen. Dieses Buch wurde z.B. vollständig auf dem V3.0-Editor geschrieben, was übrigens auch seine Kapazität andeutet. Texte bis zur jeweils maximalen Länge (bei 1 Megabyte-Speicher ca. 750 KByte) werden anstandslos verarbeitet. Ein Suchvorgang durch den gesamten Text dauerte im Durchschnitt nicht länger als 1 bis zwei Sekunden (!).

Last but not least sollen hier noch die neuen Editor-Funktionen genannt werden, wovon vor allem <Control><U> (die zuletzt durch <Control><Delete> gelöschte Zeile restaurieren) <Control>-<P>/<Control><O> (Teilzeilen löschen und restaurieren), die Funktionstasten-Belegung, die interne Zeilennumerierung und der History-Zeilenspeicher im Direktmodus hervorzuheben sind.

Man mag mir vorhalten, daß ich nichts anderes kenne als das GFA-BASIC, was vielleicht auch zum Teil stimmt, aber nach allem, was ich kenne, ist das V3.0-GFA-BASIC mitsamt seinem Editor das beste Programm, was es es für den Atari ST zu kaufen gibt. Ich bin jedenfalls restlos begeistert und ich nehme an, daß es den meisten von Ihnen ganz genauso gehen wird.

4. Basis-BASIC

Es ist unmöglich, in einem einzelnen Buch, das die Programmierung in einer bestimmten Programmiersprache erläutern soll, allen Ansprüchen gerecht zu werden. Richtet es sich nach den Interessen der Anfänger, wird es für den Fortgeschrittenen und Profi langweilig. Richtet es sich dagegen nach den Bedürfnissen der Könner, versteht der Anfänger nur noch wenig. Also muß versucht werden, einen Kompromiß zu schaffen. Dieser besteht darin, dem Anfänger die Grundlagen der Programmierung nahezubringen, ohne in Banalität zu versinken, und komplexe Sachinhalte für Fortgeschrittene darzustellen, ohne in Fach-Chinesisch abzudriften.

Um nun Anfängern die Möglichkeit zu eröffnen, mit GFA-BASIC den Grundstein zu ihrer Programmierer-Karriere zu legen, will ich hier die wesentlichen Grundlagen dieser Programmiersprache erläutern und zusätzlich eine Einführung in die Computer-Linguistik anbieten.

Wer also der Meinung ist, er sei über den Aufbau eines Computers, über Boolesche Logik, Zahlensysteme etc. bereits ausreichend informiert, kann dieses Kapitel vernachlässigen.

Den Einsteigern möchte ich allerdings empfehlen, sich hier mit dem nötigsten Rüstzeug auszustatten, denn ohne gewisse Grundkenntnisse läßt sich auch der bedienungsfreundlichste Computer nicht zu sinnvollen Betätigungen bewegen.

Dann wollen wir jetzt einsteigen: ein Computer ist in erster Linie ein äußerst dummer Zeitgenosse. Ob sich das in Zukunft mit Bio- und Megachips, Transputern, Supraleitern u.ä. wesentlich ändern wird, bleibt abzuwarten. Da Computer der gegenwärtigen Generation nur die beiden Zahlen 0 und 1 unterscheiden können, muß man manchmal gewaltige Anstrengungen unternehmen, um ihre Aufmerksamkeit zu erregen.

Unter normalen Umständen begegnet ein Laie einem Computer mit Skepsis, aber auch mit einer unleugbaren Faszination. Diese Faszination ist der Grund dafür, daß man manchmal vor lauter Ehrfurcht den eigentlich simplen Charakter eines solchen Geräts nicht erkennt. Das einzig bewundernswerte daran ist die mikroskopische Größe der Schaltungen, die fast unfaßbare Geschwindigkeit, mit welcher die verschiedenen Operationen durchgeführt werden und die genial geflochtenen Leiterbahnen auf einem fingernagelgroßen Mikrochip.

In jedem Fall sind es kreative und mit einer äußerst hohen analytischen Intelligenz begabte Menschen, die so ein Ding gebaut haben. Wenn also Ehrfurcht, dann vor den Informatikern, Technikern und Physikern, nicht vor dem Gerät. Wenn Sie nämlich die Stromzufuhr zu den Computer- Prozessoren unterbrechen, ist das Gerät Ihnen hilflos ausgeliefert. Es ist in gewisser Weise sogar sehr wichtig, sich dieses zu vergegenwärtigen, da die Chance, kreativ und produktiv mit einem Computer zu arbeiten steigt, je mehr man seine Ehrfurcht ihm gegenüber abbaut.

Nach dieser Einleitung nun zur Technik. Es ist hier nicht möglich, in die tieferen Sphären der Computertechnik einzusteigen. Deshalb will ich mich damit begnügen, Ihnen einige Begriffe zu erläutern. Dabei werde ich mich auf solche Begriffe beschränken, die Ihnen in Ihrer Programmierer-Karriere oft begegnen werden und deren Kenntnis zum Verständnis des Computer-Jargons hilfreich ist.

4.1 Computer-ABC

Jeder Computer verfügt über eine zentrale Arbeitseinheit (CPU = Central Processing Unit). Diese ist das eigentliche Herz des Computers. Es handelt sich dabei um einen Prozessor (Arbeits-Chip), der von den o.g. Informatikern so programmiert wurde, daß er selbstständig in der Lage ist, eingehende Befehle zu erkennen und diesen entsprechend zu reagieren. Befehle werden im allgemeinen über die Tastatur eingegeben oder als Programm eingelesen. Im Atari ST sind dazu die verschiedenen Schnittstellen durch ein Bündel von Leitungen mit der CPU verbunden. Dieses Leitungsbündel nennt man Bus. Es gibt Daten- ,Adreß- und Steuerbusse. Während uns der Steuerbus hier nicht näher interessieren soll, sind die Daten- und Adressbusse doch von erheblicher Bedeutung.

Der Datenbus wird dazu verwendet, Daten (Integer-Binär-Werte) zwischen den den Einheiten auszutauschen. So ist es möglich, z.B. einen Wert über die Tastatur an die CPU zu senden, die diesen dann entsprechend der gewünschten Operation verarbeitet und ggfs. im Speicher ablegt oder (was eigentlich dasselbe ist) auf dem Monitor ausgibt. Unter dem Speicher versteht man eine Ansammlung von Speicherchips, die ebenfalls durch Busse mit der CPU in Verbindung stehen. Hier kommt der Adreßbus ins Spiel. Um den gesamten Speicher organisieren zu können, wird jedem einzelnen Speicher-Byte (8-Bit-Speicherplatz) eine eigene Adresse zugewiesen. Durch Angabe dieser Adresse ist es also möglich, auf jedes einzelne Byte des Spei-

chers zuzugreifen. Zu den Bits und Bytes kommen wir später. Zunächst sehen wir uns noch einmal die Übertragungsmöglichkeiten per Bus an.

Wie gesagt, ist der Bus ein Bündel an Leitungen. Die CPU verfügt über eine Vielzahl von Pins (Steckfüsse des Chips), von denen beim ST genau 16 für Daten-Codes und 23 für Adreß-Codes verwendet werden. Diese Pins sind direkt mit den Bussen verbunden. Da in der Digital-Technik eine Stromleitung nur zwei Zustände annehmen kann (an und aus), ist es nicht möglich über eine einzelne Busleitung andere Werte als 0 (für aus) und 1 (für an) zu senden. Um einen Adreßraum von bis zu 4 Megabyte (Mega ST) zu versorgen, reicht eine Leitung also nicht aus.

Wenn man sich nun eine Stromleitung vorstellt, in die über einen Schalter Strom eingeleitet wird, und man faßt das freie Ende dieser Leitung an, bekommt man einen Schlag. Genauso geht es den Chips, die die jeweilige Information aufzunehmen haben. Aufgrund dieses elektrischen Impulses "weiß" nun der Empfänger, daß ihm etwas Bestimmtes übermittelt werden soll. Er ist darauf programmiert, entsprechend der eintreffenden Informationen entweder einen bestimmten Prozeß auszulösen, auszuführen oder die Information einfach nur zu behalten (speichern). Aber was kann man schon mit einer einzigen Leitung anfangen, die entweder die Information "Ja" (1=an) oder "Nein" (0=aus) übermitteln kann. Ein Gesprächspartner, der auf die Dauer nur Ja oder Nein sagt, wird schnell langweilig. Man möchte konkretere Auskünfte!

4.2 Bits und Bytes

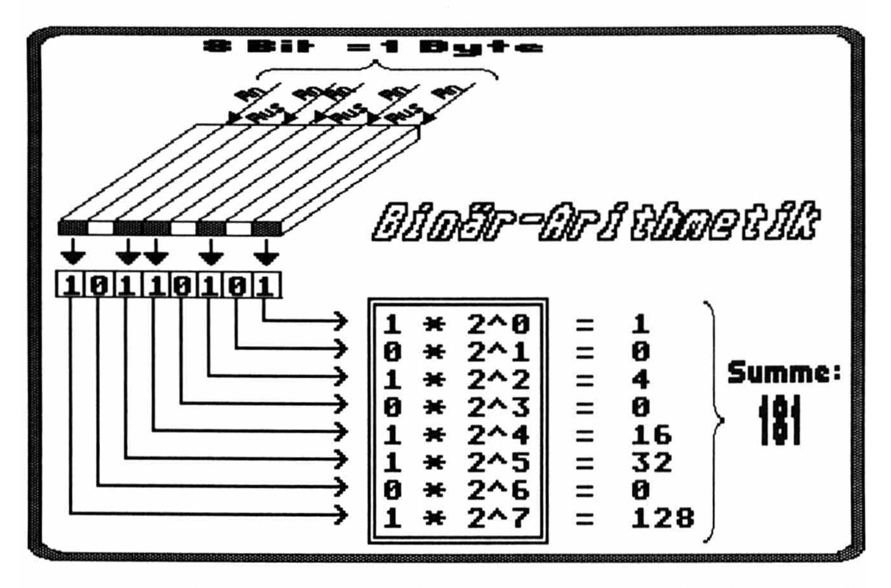
Dazu benötigen wir mehrere Informationseinheiten (Bit = engl. Abk. für Binary Digit), durch deren Kombination eine Vielfalt an unterschiedlichsten Zuständen ausgedrückt werden kann.

Nimmt man nun zwei Stromleitungen, die unabhängig voneinander an- oder ausgeschaltet werden, sind schon vier verschiedene Kombinationen denkbar. Stellen wir jede stromführende Leitung als 1 und jede "leere" Leitung als 0 dar, dann sieht das so aus:

```
00  =  Beide Leitungen führen keinen Strom
10  =  Leitung 1 =  An/Leitung 2 = Aus
01  =  Leitung 1 = Aus/Leitung 2 = An
11  =  Beide Leitungen an
```

Dies ist also schon ein kleiner Schritt mehr in Richtung Kommunikation. Um es noch deutlicher zu machen, wird das Spiel mit vier Stromleitungen wiederholt:

```
0000  =  Alle Leitungen aus
1000  =  Leitung 1 an/2, 3 und 4 aus
0100  =  Leitung 2 an/1, 3 und 4 aus
0010  =  Leitung 3 an/1, 2 und 4 aus
0001  =  Leitung 4 an/1, 2 und 3 aus
1100  =  Leitung 1 und 2 an /3 und 4 aus
0110  =  Leitung 2 und 3 an /1 und 4 aus
0011  =  Leitung 3 und 4 an /1 und 2 aus
1001  =  Leitung 1 und 4 an /2 und 3 aus
1010  =  Leitung 1 und 3 an /2 und 4 aus
0101  =  Leitung 2 und 4 an /1 und 3 aus
1110  =  Leitung 1, 2 und 3 an/4 aus
0111  =  Leitung 2, 3 und 4 an/1 aus
1011  =  Leitung 1, 3 und 4 an/2 aus
1101  =  Leitung 1, 2 und 4 an/3 aus
1111  =  Alle Leitungen an
```



Mit jeder weiteren Leitung verdoppelt sich die Anzahl der Darstellungsmöglichkeiten:

```
Alle Leitungen aus             =   0
1. Leitung an  =    2^0        =   1
2.    "    an  =    2^1        =   2
3.    "    an  =    2^2        =   4
4.    "    an  =    2^3        =   8
5.    "    an  =    2^4        =  16
6.    "    an  =    2^5        =  32
7.    "    an  =    2^6        =  64
8.    "    an  =    2^7        = 128
                              -----------
Summe aller Möglichkeiten,
die mit einem BYTE (= 8 BIT)
dargestellt werden können      = 256  (inkl. Null)
```

Ich habe diese Liste bewußt mit der Potenz 7 enden lassen, da diese 8 Leitungen mit ihren 256 verschiedenen Aussagemöglichkeiten eine Grundeinheit in der Computerlogik darstellen. Diese Einheit wird Byte genannt. Ein Computer mit einem Arbeitsspeicher von 1 Million Byte kann also 1 Million mal unabhängig voneinader einen solchen Informationsblock (Byte) von je 8 Bit aufnehmen.

Wie Ihnen vielleicht schon bekannt ist, verfügt der Atari ST über 256 verschiedene Schriftzeichen. Nach den letzten Ausführungen wissen Sie, daß diese Zahl also kein Zufall ist. Warum nun als erste Potenz eine Null gewählt wird, ist eine mathematische Festlegung. Jeder Wert der mit dem Wert Null potenziert wird, ergibt den Wert Eins. Diese Eins ist in jedem Zahlensystem die kleinste Einheit.

4.3 Binär-Arithmetik

Der Trick an der Sache ist nun der, daß jeder Zustand, der sich mit diesen 8 Leitungen darstellen läßt, auch mit einem Wert belegt werden kann. Es wurde nun ein Zahlensystem entwickelt, das ausschließlich die Zustände "An" und "Aus" zur Zahlendarstellung verwendet.

Dabei geht man mathematisch genauso vor, wie wir es von unserem Dezimalsystem her kennen. Der einzige Unterschied ist der, daß als Basis zur Potenz nicht der Wert 10, sondern der Wert 2 genommen wird. Die niedrigste Stelle einer Binärzahl (Bi = griech: Zwei) steht ebenso wie in einer Dezimalzahl rechts und die höchste Stelle links.

Als Beispiel nehmen wir ein beliebiges Byte:

```
                          01011101       =  93
Dezimal:        (10 hoch 0)   * 3        =   3
           +    (10 hoch 1)   * 9        =  90
                                      -----------
                         Ergebnis        =  93
                                      ===========
Binär   :        (2 hoch 0)   * 1        =   1
           +     (2 hoch 1)   * 0        =   0
           +     (2 hoch 2)   * 1        =   4
           +     (2 hoch 3)   * 1        =   8
           +     (2 hoch 4)   * 1        =  16
           +     (2 hoch 5)   * 0        =   0
           +     (2 hoch 6)   * 1        =  64
           +     (2 hoch 7)   * 0        =   0
                                      -----------
                         Ergebnis        =  93
                                      ===========
```

4.4 Das Hexadezimal-System

Wozu braucht man nun noch das Hexadezimalsystem? Wollte man diese Bytes als Binärzahl darstellen, müßte man dazu jedesmal eine Zeichenkette von 8 Einsen und Nullen schreiben. Um diese Zahlendarstellung zu vereinfachen, hat man sich die Hexadezimalzahlen ausgedacht.

Dieses Zahlensystem hat alle Eigenschaften der anderen Systeme. Der einzige Unterschied ist der, daß als Basis zu den Potenzen weder die 2 noch die 10 genommen wird, sondern der Wert 16. Das hat den Vorteil, daß sich die Hälfte eines Bytes (auch Tetrade oder Nibble genannt), also 4 Bit (maximal darstellbarer Wert = 15) mit einer einzigen Hexadezimalziffer darstellen läßt. Da sich jedoch mit den uns üblicherweise bekannten Zahlen keine größere Zahl als 9 einstellig schreiben läßt, mußten für die Zahlen 10 bis 15 Buchstaben gewählt werden. Der Zahl 10 wird der Buchstabe "A" zugeordnet, der Zahl 11 das "B", 12 = "C", 13 = "D", 14 = "E" und die 15 erhält den Buchstaben "F".

Somit läßt sich also die größte Zahl, die ein ST im GFA-BASIC verarbeiten kann, nämlich 2^31-1, mit nur 8 Ziffern darstellen (7FFFFFFF). Im allgemeinen werden Hexadezimalzahlen gekennzeichnet, indem ihnen ein "$" (Dollar, z.B. $1AF7) vorangestellt wird. Das GFA-BASIC handhabt dieses jedoch anders. Hier erhalten Zahlen im

Hexa-Format das Kürzel "&H" (z.B. &H1AF7). Hexzahlen haben in erster Linie den Vorteil der Zeit- und Platzersparnis.

Siehe hierzu auch unter "BIN$, HEX$, OCT$".

4.5 Codes und Opcodes

Ein Wert kann bei entsprechender Vorgabe stellvertretend als numerisches Symbol für ein bestimmtes Wort, Zeichen oder einen Befehl interpretiert werden.

Bei den Schriftzeichen (ASCII-Zeichen = American Standard Code for Information Interchange; engl.: amerikanischer Standard-Code für Informationsaustausch) steht z.B. der Wert 65 für den Buchstaben "A", der Wert 66 für "B", der Wert 67 für "C" usw. (ASCII-Tabelle siehe Anhang). Angenommen, der Wert 192 stünde für "Gebe den im folgenden Byte enthaltenen ASCII-Wert als Schriftzeichen an Cursor-Position auf dem Bildschirm aus". Die Arbeitsweise sieht dann vereinfacht so aus, daß das TOS weiß, wenn es den Wert 192 empfängt, soll es das ASCII-Zeichen auf dem Bildschirm ausgeben, das dem auf den Befehl folgenden Byte-Wert entspricht.

Der Befehl würde dann so aussehen:

```
Dezimal     : 192      66
Binär       : 11000000 10000010
Hexadezimal: C0       42
```

Es würde also das Zeichen "B" auf dem Bildschirm ausgegeben werden. Immer dann, wenn wir auf Werte stoßen, die eine Initialisierungsfunktion haben, also als Auslöser für bestimmte Prozesse dienen, haben wir es mit Opcodes (Operation-Codes) zu tun. Solche Opcodes sind nichts anderes als ein Befehl, der symbolisch durch eine Zahl repräsentiert wird. Alles, was den Prozessor interessiert, sind Zahlen, die ihm in verschiedenen Formaten als Bit-Häppchen serviert werden.

Bei der TOS-Programmierung wurden also verschiedene Funktionsabläufe vordefiniert, die nun z.B. von einem Interpreter anhand solcher Opcodes aufgerufen werden können.

Wenn Sie also in BASIC die Zeile

```
PRINT "B"
```

eingeben, wird Ihre Eingabe vom Interpreter auf die oben beschriebene Weise in das Binärformat gewandelt und dem Betriebssystem zur Weiterverarbeitung übergeben. Genausogut könnte man

```
PRINT CHR$(66)          ! (66 = ASCII-Wert für "B")
```

schreiben, nur daß das etwas umständlicher wäre.

4.6 Words und Longwords

Um nun einen Sprung von den Bytes zu den Words zu machen, braucht man keinen großen Anlauf. Als Word wird eine Informationseinheit bezeichnet, die sich statt aus 8 Bit (1 Byte) aus 16 Bit zusammensetzt. Es können nicht nur 256 verschiedene, sondern 65536 (0+2^0+2^1+2^2+2^3+2^4+2^5+2^6+2^7+2^8+2^9+2^10+2^11+2^12+2^13+2^14+2^15) verschiedene Zustände dargestellt werden.

Wie erwähnt, arbeitet der 68000er (ST-CPU) mit einer Datenbreite von 16 Bit. Es können also problemlos Werte im Bereich von 0-65535 (0 bis 2^16-1) von ihm gelesen, verarbeitet und zurückgegeben werden. Doch es gibt noch eine Steigerung. Wenn man nur Werte bis 65535 ausdrücken könnte, könnten manche nicht einmal ihre Steuererklärung damit bearbeiten. Deshalb ist der 68000er so programmiert, daß er (wenn es ihm gesagt wird) zwei 16-Bit-Words direkt nacheinander verarbeitet und dann so vorgeht, als ob es ein 32-Bit-Wert gewesen ist.

Von diesen 32 Bits werden in GFA-BASIC allerdings nur 31 Bit zur Wertedarstellung verwendet. Das letzte (höchste) Bit wird zur Kennzeichnung negativer Zahlen verwendet.

Die Zahlen, die sich damit ausdrücken lassen, dürften fast für jede erdenkliche Steuererklärung ausreichen:

```
2 hoch 31 - 1   =   2147483647
```

Diese 32-Bit-Breite wird als Longword bezeichnet. Die CPU ist nun so programmiert, daß sie zwischen diesen drei verschiedenen Datenbreiten unterscheiden kann. Ihr ist es also egal, in welchem Format Daten übergeben werden, solange man dazusagt, welches Format gemeint ist. Bei den BASIC-Befehlen finden Sie deshalb auch POKE/PEEK (Byte-Werte schreiben/lesen), DPOKE/DPEEK (Words schreiben/lesen) und LPOKE (Longwords schreiben/lesen).

4.7 Die Speicherorganisation

RAM
"Random Access Memory"- engl.: Freier Zugriffs-Speicher (der Speicherbereich, in/aus den/dem Daten geschrieben/gelesen werden können)

ROM
"Read Only Memory" - engl.: Nur-Lese-Speicher (Speicherchips, in die Daten unveränderbar eingebrannt wurden. z.B.: ROM-TOS = im ST integriertes Betriebssystem)

TOS "Tramiel Operating System" (das ST-Standard-Betriebssystem)

Der Adreßbus hat eine Breite von 23 Bit, wovon für das RAM bei einem 520ST+ oder 1040STF nur 20 Bit genutzt werden.

```
2^20-1   =   1048575
```

Besitzer eines 520ST+ oder 1040STF werden feststellen, daß der bei Systemstart am oberen Ende des Arbeitsspeichers angesiedelte Bildschirmspeicher genau 32767 (2^15-1) Byte unterhalb dieser Adresse liegt. Geben Sie bitte dazu folgende Zeile im Direktmodus ein und vergleichen Sie den erhaltenen Wert mit der Zahl 2^20-1.

```
PRINT XBIOS(2)+32767
```

XBIOS(2) ist die Anfangsadresse des Bildschirmspeichers (siehe XBIOS). Der Bildschirmspeicher ist genau 32767 Bytes groß, wovon die ersten 32000 Bytes zur Darstellung des aktuellen Bildes benötigt werden. Die restlichen 767 Bytes werden so gut wie nie benutzt. Dieser Bereich ist also völlig frei, um darin zum Beispiel kleine Maschinenprogramme oder Werte abzulegen. Über den Adressbus kann die CPU nun also 1048575 einzelne Bytes ansprechen. Diese Bytes befinden sich in dem oben bereits erwähnten Speicher und haben nichts weiter zu tun, als sich eben die Daten zu merken, die die CPU ihnen übergibt.

Die Organisiation des Arbeitsspeichers sieht nun so aus, daß das Byte mit der kleinsten Adresse vorn liegt und alle anderen Bytes sich der Reihenfolge nach daran anschließen. Der Speicher ist also eine lange Kette aus einzelnen Bytes. Um sich einiges an Verwirrung zu ersparen, ist es wichtig, sich klarzumachen, daß der Speicher keine mehrdimensionale Sache ist, sondern, wie ein gerader Strich, eindimensional. Um sich das zu vergegenwärtigen, stellen Sie sich bitte einen sehr,

sehr langen Zug vor, dessen einzelne Waggons jeweils einem Byte entsprechen und von vorn an (ab der Lokomotive) durchnumeriert wurden.

Aus dieser langen Kette von Speicherplätzen werden nun vom Betriebssystem einzelne Teile herausgeschnitten und je nach Bedarf für verschiedene Aufgaben verwendet. So belegt z.B. ein Teil des Betriebssystems den Anfang des Speichers. Andere Teile werden für die verschiedenen Anwendungen (wie z.B. den BASIC-Interpreter) zur Verfügung gestellt oder eben als Bildschirmspeicher benutzt.

Der Bildschirmspeicher stellt eine Besonderheit dar. Er wird von einem Video-Kontroll-Chip ständig in einzelne, der jeweiligen Grafik-Auflösung entsprechende Zeilen zerlegt und diese Zeilen untereinander (in mittlerer und niedriger Grafik-Auflösung auch in mehreren Ebenen "Bit-Planes") ausgegeben.

4.8 Boolesche Logik

Um nun den Computer zu einer Arbeit zu bewegen, die Ergebnisse liefert, mit denen wir etwas anfangen können, mußte ein Verfahren entwickelt werden, das die Arbeitsweise des Computers unserer eigenen angleicht. Es nützt wenig, ihn stur mathematische Aufgaben lösen zu lassen, was ja seine Lieblingsbeschäftigung ist. Man will auch, daß unter bestimmten Bedingungen Entscheidungen von ihm selbstständig getroffen werden. Sonst wäre er nichts weiter als ein besserer Taschenrechner.

Ein englischer Mathematiker namens George Boole hat sich dazu eine Form der Arithmetik ausgedacht, die daher auch Boolesche Arithmetik oder Boolesche Algebra genannt wird. Seine Idee war es, für das Grundprinzip menschlicher Entscheidungen allgemeine Regeln zu bestimmen und diese auf den Computer zu übertragen.

Wie entscheidet sich ein Mensch? Entscheidungen sind die Reaktion auf einzelne oder auch eine Folge von Bedingungen.

Man nimmt eine Situation wahr, ordnet ihre Anforderungen in ein vorhandenes Handlungsschema ein und trifft aufgrund von Übereinstimmungen oder eben auch Nichtübereinstimmungen mit dem vorhandenen Wertesystem die Entscheidung darüber, was nun zu tun ist.

Wir wissen alle, was die Worte "und" und "oder" bedeuten.

Wenn es warm ist und ich Zeit habe, werde ich baden gehen.

oder

Wenn es kalt ist oder ich nicht Zeit habe, werde ich nicht baden gehen.

Wir verknüpfen mehrere Bedingungen miteinander, um daraus zu entscheiden, was zu tun oder zu lassen ist.

Nichts anderes bewirkt die Boolesche Logik. Ein Computer ist nur ein Computer, wenn er in der Lage ist, Entscheidungen dieser oder ähnlicher Art zu fällen. Dazu hat sich Boole mehrere solcher Verknüpfungsmodi einfallen lassen.

Diese sind:

```
AND, OR, NOT, XOR, IMP, EQV
```

Wobei NOT eine Ausnahme darstellt. Hier handelt es sich nicht um eine Verknüpfung, sondern um die Umkehrung der eingehenden Information. Schauen wir uns nun an, wie sich diese Verknüpfungen auf die Behandlung der eingehenden Informationen auswirken. Dabei werden nacheinander von rechts ausgehend die Bits des ersten Operanden mit den jeweiligen, gleichrangigen Bits des zweiten Operanden verknüpft.

```
AND:
                 11011001      (Byte 1)
             AND 01101101      (Byte 2)
             -------------
                 01001001      Ergebnis
PRINT BIN$(&X11011001 AND &X01101101)
PRINT &X11011001 AND &X01101101
```

Im Ergebnis-Byte wird nur dann ein Bit gesetzt, wenn an der gleichen Stelle im ersten UND im zweiten Ursprungs-Byte ein Bit gesetzt ist.

```
OR:
                 11011001      (Byte 1)
              OR 01101101      (Byte 2)
             -------------
                 11111101      Ergebnis
PRINT BIN$(&X11011001 OR &X01101101)
PRINT &X11011001 OR &X01101101
```

Im Ergebnis-Byte wird immer dann ein Bit gesetzt, wenn entweder im ersten ODER im zweiten ODER in beiden Ursprungs-Bytes ein Bit gesetzt ist.

```
XOR:
                11011001      (Byte 1)
            XOR 01101101      (Byte 2)
           ---------------
                10110100      Ergebnis
PRINT BIN$(&X11011001 XOR &X01101101)
PRINT &X11011001 XOR &X01101101
```

Im Ergebnis-Byte wird dann ein Bit gesetzt, wenn entweder im ersten ODER im zweiten, ABER NICHT in beiden Ursprungs-Bytes ein Bit gesetzt ist. XOR = eXclusiv OR = exclusives (ausschließendes) Oder.

Bei den folgenden drei Operationen ist zu beachten, daß Vergleiche jeweils im Long-Format (32 Bit) durchgeführt werden. Da das gesamte Langwort verknüpft wird, ergeben sich bei diesen Beispielen aufgrund der Zweierkomplementierung Minuswerte (siehe unter Variablen/Boole-Variable).

```
IMP:
                          11011001      (Byte 1)
                      IMP 01101101      (Byte 2)
                     ---------------
11111111111111111111111101101111        Ergebnis
    PRINT BIN$(&X11011001 IMP &X01101101)
    PRINT &X11011001 IMP &X01101101
```

Im Ergebnis wird immer dann ein Bit gesetzt, wenn in beiden Ursprungs-Bytes ein Bit gesetzt ist oder in beiden Ursprungs-Bytes kein Bit gesetzt ist oder wenn im ersten Byte kein Bit, aber im zweiten Byte ein Bit gesetzt wird. Dies nennt man Implikation. Das Ergebnis ist immer "unwahr" (also 0), wenn im ersten Byte ein Bit gesetzt ist, aber im zweiten keins. In allen anderen Fällen ist das Ergebnis "wahr" (also 1).

```
EQV:
                          11011001      (Byte 1)
                      EQV 01101101      (Byte 2)
                     ---------------
11111111111111111111111101001011        Ergebnis
    PRINT BIN$(&X11011001 EQV &X01101101)
    PRINT &X11011001 EQV &X01101101
```

Dies ist das Gegenteil zu XOR. Im Ergebnis-Byte wird dann ein Bit gesetzt, wenn in beiden Ursprungs-Bytes entweder ein Bit gesetzt ist oder in beiden Ursprungs-Bytes kein Bit gesetzt ist. Ist in einem der

beiden Bytes ein Bit gesetzt, aber im anderen nicht, ist das Ergebnis "unwahr" (also 0).

```
NOT:
                 NOT  11011001
                 --------------
11111111111111111111111100100110        Ergebnis
    PRINT BIN$(NOT &X11011001)
    PRINT NOT &X11011001
```

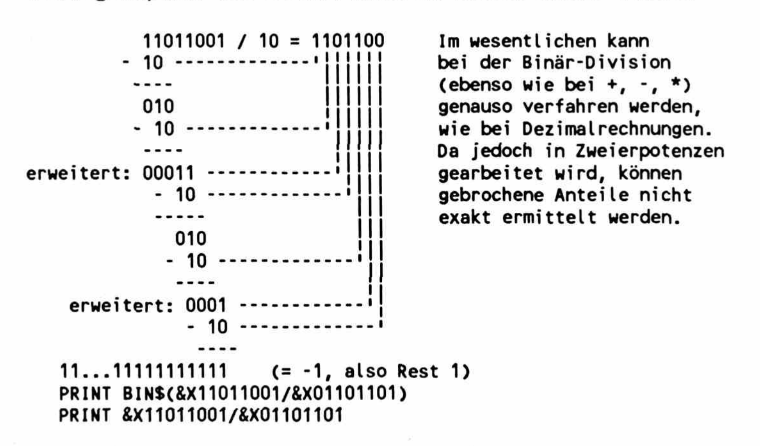
Im Ergebnis-Byte wird dann ein Bit gesetzt, wenn im Ursprungs-Byte kein Bit gesetzt ist. Das Ergebnis ist also immer das Negativ des Ursprungs-Bytes.

Mit Binärmustern lassen sich auch - wie mit Normalzahlen - Additionen, Subtraktionen, Multiplikationen und Divisionen durchführen.

```
Addition:                 11011001   (dez.=217)
                        + 01101101   (dez.=109)
                        ----------
  1.Bit 1+1          =           0   (Übertrag=1)
  2.Bit 0+0+Übertrag=           1    (Übertrag=0)
  3.Bit 0+1          =         1     (Übertrag=0)
  4.Bit 1+1          =        0      (Übertrag=1)
  5.Bit 1+0+Übertrag=        0       (Übertrag=1)
  6.Bit 0+1+Übertrag=       0        (Übertrag=1)
  7.Bit 1+1+Übertrag=      1         (Übertrag=1)
  8.Bit 1+0+Übertrag=     0          (Übertrag=1)
  9.Bit = Übertrag  =    1
                        ----------
                         101000110   (dez.=326)
  PRINT BIN$(&X11011001+&X01101101)
  PRINT &X11011001+&X01101101
Subtraktion:              11011001   (dez.=217)
                        - 01101101   (dez.=109)
                        ----------
  1.Bit 1-1          =           0   (Übertrag=0)
  2.Bit 0-0          =          0    (Übertrag=0)
  3.Bit 0-1          =         1     (Übertrag=1)
  4.Bit 1-1-Übertrag=         1      (Übertrag=1)
  5.Bit 1-0-Übertrag=        0       (Übertrag=0)
  6.Bit 0-1          =      1        (Übertrag=1)
  7.Bit 1-1-Übertrag=      1         (Übertrag=1)
  8.Bit 1-0-Übertrag=     0          (Übertrag=0)
                        ----------
                           1101100   (dez.=108)
  PRINT BIN$(&X11011001-&X01101101)
  PRINT &X11011001-&X01101101
Multiplikation:
                    11011001 * 01101101     (dez.: 217*109)
                    -------------------
  OP1/Bit1 * OP2=              01101101     (Rang: 2^0)
+ OP1/Bit2 * OP2              +00000000     (Rang: 2^1)
  Zwischenergebnis =           001101101
+ OP1/Bit3 * OP2             +00000000      (Rang: 2^2)
```

```
  Zwischenergebnis =         0001101101
+ OP1/Bit4 * OP2            +01101101        (Rang: 2^3)
  Zwischenergebnis =        01111010101
+ OP1/Bit5 * OP2           +01101101         (Rang: 2^4)
  Zwischenergebnis =        101010100101
+ OP1/Bit6 * OP2          +00000000          (Rang: 2^5)
  Zwischenergebnis =       0101010100101
+ OP1/Bit7 * OP2         +01101101           (Rang: 2^6)
  Zwischenergebnis =      10010111100101
+ OP1/Bit8 * OP2        +01101101            (Rang: 2^7)
                        ---------------------
                         101110001100101     Ergebnis
  PRINT BIN$(&X11011001*&X01101101)
  PRINT &X11011001*&X01101101
Division (Integer):
```

Die bitweise Division ist ein relativ kompliziertes Unterfangen. Aus diesem Grund wird hier nur ein einfaches Beispiel behandelt, dessen nähere Erläuterung zu weit führen würde. Es kann hier nur ein ganzzahliges Ergebnis entstehen, da die bitweise Realzahl-Division auf diese Art nicht machbar, bzw. derart aufwendig ist, daß das Thema mehrere Seiten füllen würde.



```
          11011001 / 10 = 1101100       Im wesentlichen kann
        - 10 -------------'|||||        bei der Binär-Division
        ----               |||||        (ebenso wie bei +, -, *)
          010              |||||        genauso verfahren werden,
         - 10 -------------'||||        wie bei Dezimalrechnungen.
          ----              ||||        Da jedoch in Zweierpotenzen
erweitert: 00011 ------------'|||       gearbeitet wird, können
            - 10 -------------'||       gebrochene Anteile nicht
           -----               ||       exakt ermittelt werden.
             010               ||
            - 10 -------------'|
             ----              |
     erweitert: 0001 ----------'|
               - 10 -------------'
                ----
    11...11111111111    (= -1, also Rest 1)
    PRINT BIN$(&X11011001/&X01101101)
    PRINT &X11011001/&X01101101
```

Bei all diesen Beispielen wurde das Format Byte gewählt, da Ihnen dieses Format am häufigsten begegnen wird. Grundsätzlich sind diese Operationen mit jedem Format durchführbar.

Die eben behandelte Thematik gehört nicht gerade zu den einfachen Dingen in der Computerwelt. Eine Notwendigkeit, mit logischen Operatoren und Bit-Arithmetik umzugehen, besteht allerdings nur für jene, die den Ehrgeiz haben, in die tieferen Ebenen der Computer-Programmierung hinabzusteigen. Wenn Sie dazu nicht die Neigung

verspüren, ist es auf jeden Fall gut, etwas davon gehört, bzw. gelesen zu haben.

4.9 Bedingungen und Konsequenzen

In einer anderen Beziehung ist es jedoch ausgesprochen ratsam, sich doch zumindest mit den beiden Boole-Operatoren AND und OR auseinanderzusetzen. Es gibt nämlich mehrere Befehle im BASIC, die die Angabe einer Bedingung erforderlich machen.

Nehmen wir als Beispiel den Befehl IF...ELSE...ENDIF (siehe dort). Dieses ist wohl der gebräuchlichste Befehl, um den Fortlauf des Programms von einer Bedingung abhängig zu machen. Weiter vorn wurden zwei typische Bedingungen und ihre Konsequenzen vorgestellt, wie sie in dieser oder einer ähnlichen Art im täglichen Leben ständig vorkommen.

> *Wenn es warm ist und ich Zeit habe, werde ich baden gehen.*
>
> *Wenn es kalt ist oder ich nicht Zeit habe, werde ich nicht baden gehen.*

Es werden in beiden Fällen zwei Bedingungen gestellt, deren Erfüllung mit einer Konsequenz verbunden ist.

Um das nun in ein anwendbares Beispiel zur Programm-Entscheidung übertragen zu können, setzen wir für einige Worte in den beiden Sätzen Symbole ein. Für jeden Ausdruck, der etwas bejaht, nehmen wir den Wert 1 und für jeden Ausdruck, der etwas verneint, setzen wir den Wert 0.

Also:

```
ist = 1
haben = 1
gehen = 1
nicht haben = 0
nicht gehen = 0
```

In die Struktur einer IF-Bedingung eingefügt, bekommen die beiden Sätze nun folgende Form:

```
If warm=1 And Zeit=1      ! Wenn warm "ist" UND Zeit "haben"
  Baden=1                 !  dann baden "gehen"
Endif                     ! Ende der Konsequenz
If Kalt=1 Or Zeit=0       ! Wenn kalt "ist" ODER Zeit "nicht haben"
  Baden=0                 !  dann baden "nicht gehen"
Endif                     ! Ende der Konsequenz
```

Sie merken, daß bei Verwendung von Symbolen für "Ja" und "Nein" schon recht komplizierte Entscheidungen möglich sind. Das kann man sogar noch wesentlich weiter führen, wenn man eine weitere Möglichkeit anwendet, die einem das GFA-BASIC bietet, man kann mehrere Bedingungen mit einer Klammer zusammenfassen und dazu alternativ weitere Bedingungen stellen.

Eine solche Alternative könnte im obigen Beispiel sein:

...oder wenn die Badehalle auf ist,...

Der vollständige Satz wäre dann:

Wenn es warm ist oder wenn die Badehalle auf ist, und ich Zeit habe, werde ich baden gehen.

Für die positive Eigenschaft "auf" setzen wir den Wert 1.

IF-Struktur:

```
If (Warm=1 Or Halle=1) And Zeit=1    ! Wenn (warm "ist" ODER
                                     ! Badehalle "auf")
                                     ! UND Zeit "haben"
  Baden=1                            !  dann baden "gehen"
Endif                                ! Ende der Konsequenz
```

Der Faktor "Zeit" bezieht sich hier auf beide vorangestellten Alternativen. Wenn ich z.B. auch in die Badehalle gehen würde, wenn ich keine Zeit hätte, müßte die Klammer anders gesetzt werden, da sich "Zeit" dann nur auf "warm" bezieht:

```
If (warm=1 And Zeit=1) Or Halle=1    ! Wenn (warm "ist" UND
                                     ! Zeit "haben")
                                     ! ODER Badehalle "auf"
  Baden=1                            !  dann baden "gehen"
Endif                                ! Ende der Konsequenz
```

Wenn im vorherigen Beispiel "warm" ODER "Badehalle" die Alternativen waren, so sind es jetzt "(warm UND Zeit)" ODER "Badehalle".

Zur IF-Struktur ist hier zu sagen, daß diese grundsätzlich mit einem ENDIF abgeschlossen werden muß, um dem Interpreter kenntlich zu machen, welche Konsequenzen zu welcher Bedingung gehören.

Es gibt auch noch die Möglichkeit, eine Alternativ-Konsequenz zu formulieren.

Wenn ich sage, daß ich unter bestimmten Bedingungen etwas tun werde, so folgt daraus implizit, daß ich dann, wenn die Bedingungen nicht erfüllt sind, etwas anderes tun werde.

Im obigen Beispiel könnte das sein:

Wenn es warm ist oder wenn die Badehalle auf ist, und ich Zeit habe, werde ich baden gehen.

Andernfalls werde ich nicht baden gehen und ein Buch lesen.

Für "lesen" (positiv) setzen wir hier wieder eine 1 und für "nicht lesen" (negativ) eine 0.

IF-Struktur:

```
If (Warm=1 Or Halle=1) And Zeit=1  ! Wenn (warm "ist" ODER --------.
                                   ! Badehalle "auf")              |
                                   ! UND Zeit "haben"              |
  Baden=1                          !   dann baden "gehen"          |
  Buch=0                           !   dann Buch "nicht lesen"     |
Else                               ! sonst ------------------------|
  Baden=0                          !   baden "nicht gehen"         |
  Buch=1                           !   Buch "lesen"                |
Endif                              ! Ende d. Alternativ-Konsequenz-'
```

Der Ausdruck ELSE steht hier für "andernfalls" oder auch "sonst". ELSE ist also die konsequente Umkehrung der bei IF gestellten Bedingungen. Die zwischen ELSE und ENDIF eingeschlossenen Konsequenzen bekommen nur dann Gültigkeit, wenn keine der bei IF gestellten Bedingungen zutrifft.

Das heißt wiederum, daß immer dann, wenn das Programm auf eine IF-Abfrage trifft, die mit einer ELSE-Anweisung verbunden ist, entweder die unter IF oder die unter ELSE eingebundenen Konsequenzen Gültigkeit bekommen. Soll das nicht geschehen, wird die ELSE-Anweisung einfach weggelassen. D.h., daß die Nichterfüllung der unter IF gestellten Bedingungen keine weitere Konsequenz hat, als daß das Programm hinter der zugehörigen ENDIF-Anweisung fortgesetzt wird.

Um nun nicht alle Bedingungen, die abgefragt werden sollen, in eine einzige Programmzeile schreiben zu müssen, oder um mehrere Folgebedingungen definieren zu können, können solche IF-Abfragen auch "verschachtelt" werden. Den Begriff "Verschachteln" werden Sie weiter unten auch bei den Schleifen-Strukturen wiederfinden. Damit ist gemeint, daß z.B. in einer IF-Abfrage weitere Abfragen auftreten können, so daß auch Unterverzweigungen möglich werden.

```
If Warm=1 Or Halle=1   ! Wenn warm "ist" ODER Halle "auf" --.
  If Zeit=1            !   UND Zeit "haben" ---------.       |
    Baden=1            !     dann baden "gehen"      |       |
  Else                 !   sonst --------------------|       |
    Baden=0            !     baden "nicht gehen"     |       |
  Endif                !   Ende der Alternative -----'       |
  Buch=0               !   in beiden Fällen "nicht lesen"    |
Else                   ! sonst   ----------------------------|
  Buch=1               !   Buch "lesen"                      |
Endif                  ! Ende d. 1. Alternativ-Konsequenz --'
```

Dieses Beispiel entscheidet in zwei Stufen, welche Konsequenzen die Erfüllung zweier unabhängiger Bedingungen haben soll. Erst wenn es warm ist oder die Badehalle auf ist, soll der Zeitfaktor in Betracht gezogen werden. Ist es weder warm, noch ist die Badehalle auf, so wird der Zeitfaktor von vorneherein vernachlässigt und die Entscheidung "Buch lesen" getroffen.

Anhand dieser einfachen Beispiele ist die Übertragbarkeit alltäglicher Entscheidungen in die Logik der Computerwelt hoffentlich etwas deutlich geworden. Mit Zunahme Ihrer Routine wird auch die Einsicht in die Möglichkeiten dieser Verknüpfungen wachsen. Es ist jedenfalls manchmal recht faszinierend, wie sich durch komplexe Bedingungen unterschiedliche Einflüsse so abfangen und verarbeiten lassen, daß schon der Eindruck eines "intelligenten" Programms entsteht. Die Virtuosität im Umgang mit Bedingungen kennzeichnet den guten Programmierer.

4.10 Flags

Oben wurde noch ein weiteres Prinzip effektiver Programmierung sichtbar. Man nennt es Flags (engl.: Flaggen).

Diese Flaggen haben eine sehr wichtige Funktion in jedem Programm, das nicht nur die Grundfunktionen und -strukturen verwendet, sondern darüber hinaus verschiedene Zustände signalisieren kann, die dann in die Entscheidungsfindung einbezogen werden sollen. Bemühen wir noch einmal unser Beispiel:

```
If Warm=1 Or Badehalle=1
  Flag=1
Endif
.
. ... weiteres Programm ...
.
If Flag=1 And Zeit=1
  Baden=1
Endif
```

Es kann also ein Zustand an irgendeiner Programmstelle ausgewertet werden, dessen Ergebnis erst später zur Wirkung kommen soll. Es ist hier denkbar, daß die in "Flag" gespeicherte Information an mehreren Stellen im Programm ausschlaggebend sein soll.

Um nun nicht an jeder dieser Stellen die Entscheidung treffen (und auch definieren) zu müssen, ob es warm ist oder ob die Badehalle auf ist, kann man diese Entscheidung bei frühestmöglicher Gelegenheit vornehmen und die Information, ob die Entscheidung positiv oder negativ ausgefallen ist, in einer Variablen speichern und diese dann bei weiteren Gelegenheiten abfragen.

Vielleicht weiß ich heute schon, daß es morgen warm sein wird, aber ich weiß heute noch nicht, ob ich morgen Zeit haben werde, baden zu gehen. Also treffe ich die zweite Teilentscheidung erst dann, wenn die dazu erforderlichen Umstände eingetreten sind. Im Beispiel wurde der Variablenname "Flag" willkürlich gewählt. Sie können dafür natürlich jeden beliebigen Variablennamen verwenden.

4.11 Die Variablen

Variablen haben *in einem Programm dieselbe Funktion*, wie wir sie aus der Mathematik kennen. Sie werden als Platzhalter für Größen oder Ausdrücke (numerische oder alphanumerische) eingesetzt, deren Inhalte erst im Programmverlauf ermittelt und zugewiesen werden und sich im weiteren Programm ständig verändern können. Wir wissen also entweder zum Zeitpunkt der Programmentwicklung nur, "daß" etwas in diesen Variablen abgelegt wird, aber noch nicht "was", oder wir weisen ihnen im Programm-Listing Inhalte zu, die wir an den gegebenen Stellen für notwendig halten.

Um keine Mißverständnisse aufkommen zu lassen: auf eine Art wissen wir im ersten Fall schon, "was" diese Variablen aufzunehmen haben, wir wissen nur nicht, welcher konkrete Inhalt es sein wird. Denn eine Entscheidung hat man von vorneherein selbst zu treffen. Nämlich,

welcher Variablentyp einzusetzen ist. Es gibt zwei grundlegend verschiedene Typen von Variablen. Das eine sind die numerischen, bzw. Werte-Variablen und das andere die alphanumerischen, bzw. Text- oder auch String-Variablen.

Numerische Variablen haben die Aufgabe, Werte zu speichern.

```
Hypo=SQR(22^2+13^2)  ! Hypo = Wurzel aus (22 hoch 2 + 13 hoch 2)
```

Ein Beispiel, das wohl alle aus der Schule kennen. Es werden die beiden Kathetenlängen eines rechtwinkligen Dreiecks quadriert und die Quadrate addiert. Anschließend wird nach dem Satz des Pythagoras die Länge der Hypothenuse berechnet. Das Ergebnis dieser Berechnung wird nun der Variablen "Hypo" zugewiesen. Solange keine weiteren Werte an diese Variable übergeben werden, enthält sie nun also die Länge der Hypothenuse des angegebenen Dreiecks. Dieser Wert kann im Laufe des Programms beliebig oft erfragt oder auch durch Neuzuweisungen verändert werden.

Es gibt nun wieder drei verschiedene Typen (in V3.0 fünf) von numerischen Variablen. Wenn wir eine Zahl speichern wollen, die auch Nachkommastellen beinhaltet, also eine sogenannte Realzahl, wird nur der reine Variablenname angegeben. Diesen Typ haben wir im obigen Beispiel kennengelernt. Er benötigt zur Speicherung der ihm übergebenen Werte generell einen Speicherplatz von 6 Byte (in V3.0 8) pro Variable. Dadurch ist eine Genauigkeit von bis zu 11 (13) Stellen möglich. Wertezuweisungen, die über diese 11 (in V3.0 13) Stellen hinausgehen, werden automatisch auf die 11. (13.) Stelle gerundet:

```
A=123.125237667231  ergibt  A=123.12523767
```

Bei ganzzahligen Anteilen von mehr als 11 Stellen (in V3.0 13) wird die übergebene Zahl automatisch in das Exponentialformat gewandelt:

```
A=642653017623.527  ergibt  A=6.4265301762E+11
```

Andererseits werden Wertzuweisungen, die als "Normal"-Zahl darstellbar sind und im Exponentialformat angegeben wurden, in das Normalformat gewandelt.

```
A=1284.55E+5        ergibt  A=128455000
```

Im Exponentialformat sind Wertzuweisungen im Bereich von -1.3407807929E+154 bis 1.3407807929E+154 möglich.

Eine Exponentialzahl ist folgendermaßen zu lesen:

```
54.6341E+7        entspricht  54.6341 * (10 ^ 7)
```

Ein weiterer Typ ist die Integerzahl. Dies sind Zahlen, die keine Nachkommastellen beinhalten sollen. Es können ihnen also nur Ganzzahlen zugewiesen werden. Um wieder einem Irrtum vorzubeugen: es können natürlich auch Realzahlen zugewiesen werden. Nur, diese werden in einer Integervariablen nicht als solche gespeichert, sondern die evtl. mit übergebenen Nachkommastellen werden einfach "vergessen".

```
A%=149.523
PRINT A%
==========> Ausgabe:   149
```

Um dem Interpreter klar zu machen, daß er es hier mit einer Variablen des Integertyps zu tun hat, muß dem Variablennamen ein '%' (z.B. Var%) angehängt werden. Pro Variable benötigt dieser Typ einen Speicherplatz von 4 Bytes, woraus sich ein Integer-Bereich von -2147483648 bis 2147483647 ergibt. In V3.0 gibt es noch die 1- und 2-Byte-Integervariable.

Der dritte numerische Typ ist die Boole-Variable. In ihr können ausschließlich zwei Werte abgelegt werden. Wenn Sie sich später mit den einzelnen BASIC-Befehlen befassen werden, werden Ihnen mehrere Funktionen begegnen, die als Ergebnis ebenfalls nur zwei verschiedene Werte liefern können. Der eine Wert ist die Null. Dieser Wert gilt im Interpreter als der Wahrheitswert 0. Auch wenn er Wahrheitswert genannt wird, ist dieser Wert stellvertretend für die Feststellung "falsch". Der andere Wahrheitswert ist die Zahl -1. Dieser Wert steht grundsätzlich stellvertretend für die Feststellung "richtig".

Warum für die Feststellung "richtig" eine -1 steht, hat seinen Grund in der sogenannten Zweierkomplement-Darstellung. In einem Longword sind in diesem Fall alle Bits "gesetzt" (angeschaltet), also auch das höchste. Daraus ergibt sich ein Minuswert. Dieses Verfahren hier zu erläutern, würde den gesetzten Rahmen sprengen. Als BASIC-Anfänger muß es sie im allgemeinen auch nicht näher interessieren. Kluge Köpfe werden evtl. weiter oben bei Words und Longwords schon stutzig, wo als maximaler Wertebereich 2^31-1 genannt wurde. Dieses liegt eben an der Zweierkomplementierung, die das oberste Bit als Minus-Identifikator verwendet. Sobald vor einer Zahl ein Minuszeichen steht, wird das oberste Bit eines Longwords gesetzt und die Zahl vorzeichenlos von 2^31 abgezogen. Das daraus entstehende Bit-Muster

wird bei Zweierkomlementdarstellung als Minuswert interpretiert und dementsprechend zurückgegeben.

```
PRINT BIN$(1)
========> Ausgabe: 1                                (1 Bit)
PRINT BIN$(2^31-1)
========> Ausgabe:  1111111111111111111111111111111 (31 Bit)
PRINT BIN$(-2^31)
========> Ausgabe: 10000000000000000000000000000000 (32 Bit)
PRINT BIN$(-2^31+1)
========> Ausgabe: 10000000000000000000000000000001 (32 Bit)
PRINT BIN$(-2^31+2)
========> Ausgabe: 10000000000000000000000000000010 (32 Bit)
PRINT BIN$(-2^31+2^15)
========> Ausgabe: 10000000000000001000000000000000 (32 Bit)
PRINT BIN$(2^31-2^15)
========> Ausgabe:  1111111111111111000000000000000 (31 Bit)
A%=-1
PRINT LPEEK(VARPTR(A%))
========> Ausgabe: 11111111111111111111111111111111 (32 Bit)
```

Zurück zu den Wahrheitswerten. Nehmen wir dazu als Beispiel den Befehl EXIST. Dieser hat die (oberflächlich gesehen) einfache Aufgabe, festzustellen, ob eine bestimmte Datei auf der Diskette existiert oder nicht. Existiert die Datei, liefert EXIST den Wert -1. Andernfalls wird der Wert 0 zurückgegeben. Andere Werte können von dieser Funktion nicht geliefert werden, weil eben nur zwei Zustände auftreten können. Entweder die Datei existiert oder sie existiert nicht. Daraus ist der eigentliche Sinn der Boole-Variablen erkennbar. Überall dort, wo aus einer Entscheidungsfindung nur die Antworten "Ja" oder "Nein", bzw. "richtig" oder "falsch" resultieren kann, können Wahrheitswerte in ihnen abgelegt werden.

Die Boolevariable kann nur einen dieser beiden Werte aufnehmen. Selbst wenn Sie irgend einen beliebigen Wert zuordnen, wird der Interpreter immer nur zwischen "falsch" (0) und "richtig" (-1) unterscheiden. Alle Werte, die diesem Variablentyp übergeben werden und ungleich 0 sind, werden automatisch als 'wahr', also -1 interpretiert und abgespeichert. Dieser Variablentyp hat den Vorteil, daß er zur Speicherung seiner Inhalte nur 2 Byte pro Variable benötigt. Will man eine Variable als Boolevariable verstanden wissen, muß man dem Variablennamen das Zeichen '!' (z.B. Var!) anhängen.

In Text-, bzw. String-Variablen (String; engl.: Kette/Reihe/Schnur/Saite) werden dagegen keine Werte, sondern Textzeichen abgelegt. Genaugenommen sind diese Zeichen ebenfalls Werte, wie wir es weiter vorn schon kennengelernt haben (ASCII-Zeichen). Nur bei dieser Art der Variablen "weiß" das BASIC, daß es die hier abgelegten Werte

nicht als Zahlen, sondern als ASCII-Zeichen zu interpretieren hat. Vorausgesetzt, es wurde ihm klargemacht, daß es sich hier um eine String-Variable handelt. Das macht man, indem man dem Variablennamen ein '$' (Dollarzeichen - z.B. Var$) anhängt.

Eine String-Variable kann im GFA-BASIC eine Zeichenkette mit einer Anzahl von 0 bis 32767 einzelner Textzeichen aufnehmen. Das heißt nun nicht, daß jede String-Variable einen Speicherplatz von 32767 Byte (1 ASCII-Zeichen = 1 Byte) reserviert, sondern daß ein String mit maximal 32767 Zeichen übergeben werden kann. Die Länge, die eine solche Variable annimmt, hängt jeweils eben davon ab, wieviel Zeichen zugeordnet wurden. An Speicherplatz benötigt eine String-Variable soviel Byte, wie Zeichen vorhanden sind. Zusätzlich werden zu jeder String-Variablen noch 6 Byte für einen Pointer (engl.: Zeiger) benötigt.

Zu jeder String-Variablen existiert nämlich ein sogenannter Descriptor (engl.: Beschreiber), der sich selbstständig die Adresse, also den Standort der Variablen im Speicher, sowie ihre Länge "merkt" (mehr dazu unter ARRPTR und "Variablenorganisation/-typen").

```
A$="BASIC"
PRINT "Der String hat eine Länge von ";LEN(A$);" Zeichen"
PRINT "Die Stringadresse ist ";VARPTR(A$)
PRINT "Der Descriptor für A$ steht bei Adresse ";ARRPTR(A$)
PRINT "Das erste Byte des Strings hat den Wert ";PEEK(Varptr(A$))
PRINT "Der Wert ";PEEK(VARPTR(A$));" repräsentiert das Zeichen ";
PRINT CHR$(PEEK(VARPTR(A$)))
```

Als erstes wurde hier der Variablen "A$" der String "BASIC" übergeben. Anschließend wird mit der BASIC-Funktion LEN die Länge des Strings ermittelt. Um nun in Erfahrung zu bringen, wo das erste Zeichen (Byte) dieses Textausdrucks im Speicher zu finden ist, kann mit der BASIC-Funktion VARPTR (was soviel wie "Variablenzeiger" heißt), die Adresse erfragt werden. Mit der Speicherlese-Funktion PEEK wird nun der Byte-Wert des ersten Zeichens aus der mit VARPTR ermittelten Adresse ausgelesen. Und zum Schluß wandelt die Textfunktion CHR$ den so gelesenen Wert wieder zurück in ein Textzeichen, das genau der erste Buchstabe des übergebenen Strings ist.

Zählen Sie zu der mit VARPTR ermittelten Adresse eine 1 hinzu, haben Sie die Adresse des zweiten Zeichens. Addieren Sie eine 2, erhalten Sie die Adresse des dritten Zeichens usw.

```
A$="BASIC"
PRINT "Das zweite Zeichen hat den ASCII-Wert ";PEEK(VARPTR(A$)+1)
```

Wenn Sie den ganzen Variableninhalt auf dem Bildschirm sehen wollen, geben Sie

```
PRINT A$
```

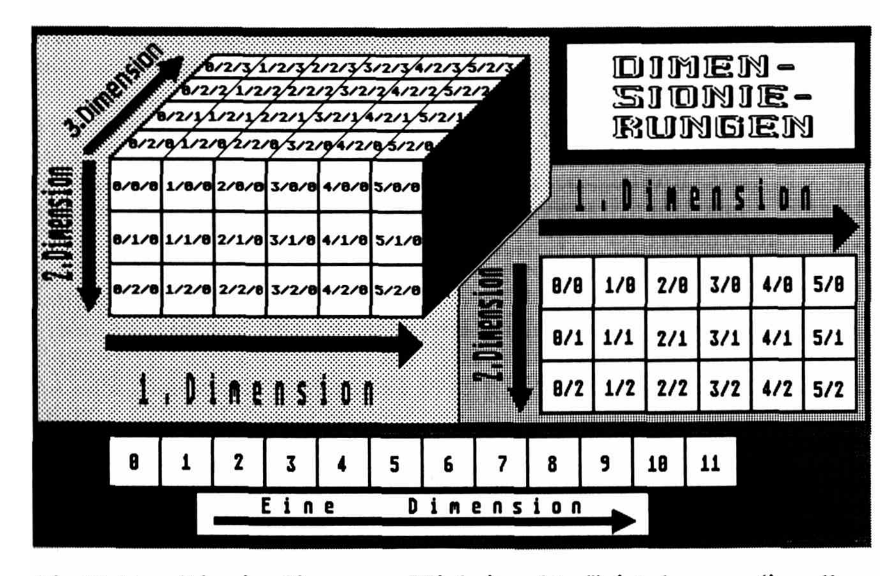
ein und der String wird auf dem Bildschirm ausgegeben.

Probieren geht über studieren! Zum obengenannten Thema finden Sie weitere Informationen unter 24.2. "Variablenorganisation/-typen".

4.12 Matrix und Vektor

Das hört sich fast an, wie der Titel eines Shakespeare-Dramas. Da es ja nicht langweilig werden soll und man außerdem ohne sie nicht auskommen kann, gibt es noch eine weitere Gattung der Variablentypen. Man nennt sie Felder oder Arrays (array = engl.: Aufstellung/Reihe/Ordnung). Wer in der Schule gut aufgepaßt hat, weiß, daß man zur Berechnung einer Funktionskurve mindestens zwei Größen benötigt. In den meisten Fällen werden dies die Größen "X" und "Y" gewesen sein. Der Berechnungsvorgang ist der, daß zu jeder angenommenen Größe "X" anhand einer Funktionsgleichung die Größe "Y" zu ermitteln war. Aus den Schnittpunkten dieser beiden Größen ergaben sich dann die Punkte der Kurve.

Diese beiden Werte stellten auf die jeweilige Funktion bezogen ein Koordinatenpaar dar. Um nun mit den jeweils zusammengehörenden Ordinaten-Werten nicht durcheinanderzukommen, kann man sich ein Feld einrichten. Dieser Vorgang ist nichts anderes, als das, was wohl die meisten unter dem Begriff Wertetabelle kennen. Solch ein zweidimensionales Feld wird auch als Matrix bezeichnet, wovon jede einzelne Dimension einen Vektor darstellt.



Ein Vektor (hier im Sinne von "Einheitsvektor") ist dagegen (im allgemeinen Verständnis) ein Feld, das nur eine Dimension besitzt und meist dazu verwendet wird, um mehrere Werte, die zu einer bestimmten Gruppe gehören, unter einer gemeinsamen "Überschrift" (dem Variablennamen) zusammenfassen und ordnen zu können.

Weitere Informationen zum Umgang mit Feldern finden Sie unter DIM bzw. "Aufbau eines mehrdimensionalen Feldes".

Außer den Boole-Variablen benötigen alle anderen Variablentypen in einem Array denselben Speicherplatz, den sie auch als Einzelvariable beanspruchen. Die Boole-Variable benötigt dagegen in einem Array pro Element nur einen Speicherplatz von einem einzigen Bit(!).

4.13 Erkennungsdienst

Mir ist immer wieder aufgefallen (auch an mir selber), daß eines der größten Probleme, eine Computersprache zu erlernen, darin besteht, daß man am Anfang in einem Listing nicht unterscheiden kann, was denn nun Befehle (also feststehende Begriffe) und was Namen (also frei bestimmbare Begriffe) sind. Dazu nun einige Grundregeln.

Die erste: Lernen Sie alle Befehlsnamen so schnell wie möglich auswendig. Alles andere können nur noch freie Begriffe sein! Vorsicht: Ironie, aber etwas Wahres ist an diesem banalen Satz.

Wenn das so einfach wäre, wie es sich anhört. Gerade bei einer Sprache wie GFA-BASIC, die in der V3.0-Version ca. 450 verschiedene Befehle, Funktionen, reservierte Variablen und Felder kennt, ist man schnell überfordert.

In GFA-BASIC kann man ab der Version 2.02 problemlos auch Befehlsnamen als Variablennamen verwenden. Es gibt also bis auf reservierte Variablen (TIMER, GINTIN, DATE$ etc.) keine reservierten Begriffe mehr. Bei Prozeduren war es auch schon vorher möglich, zur Namensbildung Befehlsnamen zu verwenden, jedoch lassen sich diese relativ einfach von Befehlen unterscheiden. Eine Prozedur beginnt immer mit der Kennung PROCEDURE, eine Funktion immer mit der Kennung DEFFN und ein Label steht immer allein, bzw. evtl. mit einem Kommentar (!Kommentar) versehen und endet mit einem Doppelpunkt.

```
Data_label:
```

oder

```
Xyz_label.1: !Kommentar abc....xyz
```

Prozedur-Aufrufe sind an dem vorangestellten GOSUB oder @ zu erkennen, während Funktions-Aufrufe immer mit FN oder (ebenfalls) mit @ beginnen.

```
Gosub Proc1        oder    @Proc1
Xy%=Fn Funk1       oder    Xy%=@Funk1
```

Bei Labels sind nur solche Bezeichnungen möglich, die vom Interpreter nicht falsch verstanden werden können. Z.B. wird ein Label mit dem Namen Save: in den Befehl SAVE ":" oder der Name Fileselect: in FILES "elect:" umgewandelt.

Der Label-Name Print: ist z.B. also ohne weiteres möglich, sollte jedoch zugunsten der besseren Überschaubarkeit unterbleiben.

Wie vorn schon ausgeführt, haben Variablen (bis auf Real-Variablen) alle eine Endkennung. Integervariablen erhalten ein '%' (Var%), Boole-Variablen ein '!' (Var!) und String-Variablen ein '$' (Var$). Diese sind also ebenfalls an ihren Kennungen leicht auszumachen.

Als Namen können beliebig lange Bezeichnungen eingesetzt werden, die sich aus den normalen Textzeichen (A-Z/a-z/0-9), sowie dem Tiefstrich _ und dem Punkt . zusammensetzen können. Bei Namen von Variablen und Funktionen muß das erste Zeichen allerdings ein Buchstabe sein.

```
V.aria_blen_name.1
Feld_titel.xyz%(Dim1,Dim2,...)
1:      <- Soll ein Label sein!
PROCEDURE 1724_von_a.bis.z
DEFFN hardcopy=X%*Y%
```

Um allen Irritationen aus dem Wege zu gehen, können Sie bei Programmen, die Ihnen nicht nur auf dem Papier begegnen, sondern die Sie in den BASIC-Editor laden und dort listen können (.BAS- und .LST-Files) einen Komfort-Befehl des GFA-BASIC verwenden. Drücken Sie, nachdem Sie das Programm geladen haben, die <Esc>-Taste. Geben Sie nun im Direktmodus

```
OK >deflist 0
```

ein und bestätigen Sie die Eingabe mit <Return>. Nachdem Sie zum Editor zurückgefunden haben (<Esc> <Return>, Eingabe von 'ed' und <Return> bzw. Drücken von <Control>+<Shift>+<Alternate>) werden Sie feststellen, daß sich die Darstellung des Listings geändert hat. Es werden nun **alle** Befehlsnamen in Großbuchstaben und **alle** anderen Namen in Kleinbuchstaben dargestellt. Für den Anfänger ist dies eine gewaltige Hilfe.

Wenn Sie sich einige Zeit mit GFA-BASIC beschäftigt haben, werden Sie diese Hilfe sicher nicht mehr benötigen, da sich allein aus der Logik der Syntax schon eine eindeutige Bestimmung ergibt. Ein Name, hinter dem ein Gleichheitszeichen steht, kann z.B. **nur** eine Variable sein und ein Name, dem ein "@" vorangestellt ist und der direkt am Zeilenanfang steht, kann **nur** ein Prozeduraufruf sein. Befindet sich dagegen vor einem Namen mit vorangestelltem "@" ein Gleichheitszeichen, kann es sich **nur** um einen Funktionsaufruf handeln usw.

Ich komme wieder auf den obigen "Spruch" zurück. Setzen Sie sich am Anfang zuerst mit den Grundlagen-Befehlen (PRINT, INPUT, READ, DATA, PEEK, POKE, GOSUB etc.) auseinander und versuchen Sie, die übrigen Komfort-Befehle und -Funktionen erst einmal weitestgehend zu ignorieren. Wenn Sie in den Grund-Befehlen sattelfest sind, erweitern Sie Ihren Sprachschatz nach und nach um die restlichen Befehle.

4.14 Schleifenstrukturen

Eine Schleife nennt man jede Form von Programmstruktur, die bewirkt, daß ein ganz bestimmter Programmblock mehrmals nacheinander durchlaufen wird. In GFA-BASIC sind vier solcher Loops (Loop = engl.: Schleife) verwendbar.

1. FOR...NEXT-Schleife
2. DO...LOOP-Schleife
3. REPEAT...UNTIL-Schleife
4. WHILE...WEND-Schleife

Untersuchen wir als erstes zwei Typen, die Bedingungsabfragen implizit verwenden.

Die REPEAT...UNTIL-Schleife

Der Schleifendurchlauf wird durch die Anweisung REPEAT eingeleitet. Im Anschluß an diese Anweisung folgt nun ein beliebig großer Programmblock, der wiederholt ausgeführt werden soll. Die Eigenart dieser Schleife ist eine Bedingungsabfrage am Ende, also am Wendepunkt der Schleife. Die dort gestellte Bedingung bestimmt, wie oft die Schleife durchlaufen werden soll, bzw. unter welchen Bedingungen die Schleife nicht mehr durchlaufen werden soll. Der Schleifenwendepunkt heißt hier UNTIL. Dieser Umkehr-Anweisung wird die genannte Bedingung beigestellt.

```
REPEAT
  INC A
  B=SQR(A)
  PRINT "Wurzel aus ";A;" = ";B
UNTIL B=15 OR A=200
```

Innerhalb der Schleife wird hier ein Zähler (A) durch INC bei jedem Durchlauf um 1 erhöht, anschließend die Wurzel daraus ermittelt und die beiden Werte ausgegeben. Wie Sie sehen, bestimmt die Bedingung B=15, daß die Schleife so oft durchlaufen wird, bis der Wurzelwert mit der Zahl 15 identisch ist. Wie auch bei IF-Abfragen können hier die Bedingungen mit logischen Operatoren verknüpft werden. So wird hier die Schleife auch (unabhängig von B) verlassen, wenn der Zähler den Wert 200 erreicht.

Die wesentliche Eigenart dieses Schleifentyps ist, daß der Schleifeninhalt auf jeden Fall mindestens einmal durchlaufen wird, da die Bedingungsabfrage erst am Ende der Schleife erfolgt.

Anders ist es bei der WHILE...WEND-Schleife. Diese wird dagegen gar nicht durchlaufen, wenn die Abbruchbedingung bereits bei Erreichen der Schleife erfüllt ist.

```
A=11
WHILE A<10
  INC A
  PRINT SQR(A)
WEND
```

Der Programmblock innerhalb der Schleife wird nicht ausgeführt. Der Schleifeneinstieg WHILE (engl.: während/solange) sagt aus, daß der Block solange durchlaufen werden soll, wie A kleiner als 10 ist. Da A bereits vorher größer ist, wird das Programm sofort hinter dem Wendepunkt WEND fortgesetzt.

Für den Fall, daß die Schleife durchlaufen wird, wird bei jedem Durchlauf geprüft, ob die bei WHILE gestellte Bedingung erfüllt ist. Ist sie das, wird der Block nicht noch einmal ausgeführt und das Programm ebenfalls hinter WEND fortgesetzt. Auch hier können, bei Angabe mehrerer Bedingungen, diese logisch verknüpft werden.

Eine sehr gebräuchliche Schleifenform, die FOR...NEXT-Schleife, verwendet dagegen keine Bedingung dieser Art, sondern führt die Schleife so oft aus, wie in einer Zählanweisung vorgegeben wird.

```
FOR A%=1 TO 225
  B=SQR(A%)
  PRINT "Wurzel aus ";A%;" = ";B
NEXT A%
```

In der FOR-Zeile wird eine beliebige Zählvariable (hier A%) angegeben, die im Verlauf der Schleife solange um den Wert 1 (hier beginnend mit 1) erhöht wird, bis sie den Endwert (hier 225) erreicht hat. Der Schleifenwendepunkt wird durch die Anweisung NEXT gekennzeichnet. Dieser Anweisung ist der Name der verwendeten Zählvariablen beizustellen. Die FOR...NEXT-Schleife verfügt noch über einige Varianten, die Sie aus der Befehlsbeschreibung zu FOR...NEXT entnehmen können.

Eine Schleife ohne Bedingungsabfrage oder Zähler ist die DO...LOOP-Schleife. Dieser Schleifentyp führt den zwischen DO und LOOP eingeschlossenen Programmblock unendlich lange aus. Es kann keine Abbruchbedingung implizit definiert werden.

```
DO
  INC A%
  B=SQR(A%)
  PRINT "Wurzel aus ";A%;" = ";B
LOOP
```

Soll eine DO...LOOP-Schleife abgebrochen werden, hat man nur die Möglichkeit, entweder die Tasten-Kombination <Control/Shift/ Alternate> zu drücken oder eine spezielle Abbruchbedingung zu stellen. Diese Abbruchbedingung heißt EXIT IF (siehe dort).

Bei allen Schleifenarten ist es möglich, diese ineinander zu verschachteln. Es kann also in einer Schleife eine weitere, gleich welcher Art, aufgerufen werden.

```
WHILE I%<10
  INC I%
  FOR J=1 TO 10
    REPEAT
      INC K
      PRINT "I% = ";I%;" J = ";J;" K = ";K
    UNTIL K>I%*10
  NEXT J
WEND
```

Bei Verschachtelungen dieser Art ist darauf zu achten, daß die jeweiligen Schleifenwendepunkte (NEXT/UNTIL/LOOP/WEND) in der umgekehrten Reihenfolge ihrer Startanweisungen (FOR/REPEAT/ DO/WHILE) gesetzt werden.

```
Falsch:                          Richtig:

      REPEAT                           REPEAT
        WHILE A<10                       WHILE A<10
        UNTIL A=10                       WEND
      WEND                             UNTIL A=10

Falsch:                          Richtig:

      FOR I=1 To 10                    FOR I=1 To 10
        FOR J=1 To 10                    FOR J=1 To 10
        NEXT I                           NEXT J
      NEXT J                           NEXT I
```

Sollten Sie diese Reihenfolge nicht einhalten, wird Ihnen der Interpreter bei Programmstart einen "Schwarzen Peter" überreichen.

Diejenigen, die es von anderen BASIC-Interpretern her gewohnt sind, Schleifen anhand von GOTO-Anweisungen zu konstruieren, sollten sich frühzeitig angewöhnen, dafür eine der obigen Schleifen-Kon-

sich frühzeitig angewöhnen, dafür eine der obigen Schleifen-Konstruktionen einzusetzen. Dieses hat einen wesentlichen Vorteil. GOTO-Anweisungen werden in GFA- BASIC nicht als Struktur-Elemente anerkannt. D.h. also, daß Schleifen nicht auf Anhieb erkennbar wären, während man GFA-Loops sofort an der Zeileneinrückung erkennen kann.

Standard-BASIC :

```
10 A=A+1:Print A;:If A<20 Then Goto 10
```

GFA-BASIC:

```
Label:           ! In diesem Fall würde man
INC A            ! natürlich eine FOR...NEXT-
PRINT A;         ! Schleife verwenden. Hier
IF A<20          ! sollen jedoch nur die
  GOTO Label     ! verschiedenen Strukturen
ENDIF            ! von Schleifen aufgezeigt werden.
```

Besser:

```
WHILE A<20     oder   DO                  oder   REPEAT
  INC A                 INC A                      INC A
  PRINT A;              PRINT A;                   PRINT A;
WEND                    EXIT IF A=>20            UNTIL A=>20
                      LOOP
```

4.15 Vergleichsoperationen

Bei der vorangegangenen Beschreibung der Schleifenstrukturen wurden mehrfach sogenannte Vergleichsoperatoren verwendet.

=	Gleich
==	Ungefähr gleich (25-Bit-Vergleich)
<	Kleiner
>	Größer
<> bzw. ><	Ungleich
<= bzw. =<	Kleiner oder gleich
>= bzw. =>	Größer oder gleich

Diese Operatoren können also eingesetzt werden, um zwei Werte oder Textausdrücke miteinander zu vergleichen.

```
PRINT "ABC">"BCD"
```

oder

```
A$="eins"
B$="zwei"
PRINT A$<>B$
```

oder

```
PRINT 123=234
```

oder

```
A%=17356
B=651423.241
PRINT A%<B
```

Diese Beispiele wirken etwas seltsam, erhalten jedoch dann einen Sinn, wenn man weiß, daß als Ergebnis eines Vergleichs immer ein Wahrheitswert (0 oder -1) geliefert wird. Die Beispiele 1 und 3 würden demnach eine Null (falsch) als Ergebnis ausgegeben, da der String ABC nicht größer ist, als der String BCD und auch der Wert 234 nicht gleich dem Wert 123 ist. Die Beispiele 2 und 4 liefern dagegen eine -1 (wahr), da A$ und B$ tatsächlich ungleich (<>) und "A%" kleiner als "B" ist.

Auch hier ist eine sinnvolle Anwendung der Boole-Variablen denkbar, indem man die ermittelten Wahrheitswerte an eine solche Variable übergibt.

Sie werden sich evtl. fragen, wie man denn Textausdrücke auf "größer" oder "kleiner" prüfen kann. Das geht folgendermaßen vor sich: Sollen zwei Textausdrücke verglichen werden, geht das BASIC der Reihe nach alle Zeichen der beiden Ausdrücke durch und ermittelt, welches der beiden verglichenen Zeichen den größeren ASCII-Wert besitzt. Der Ausdruck, der in einer der verglichenen Positionen das Zeichen mit dem größeren ASCII-Wert enthält, ist somit der "größere" String. Beim Vergleich zweier Strings auf "kleiner" wird das gleiche Verfahren angewandt. Nur ist hier eben der "kleinere" String der, der zuerst ein Zeichen enthält, dessen ASCII-Wert kleiner ist, als der des verglichenen Zeichens des anderen Strings.

Haben die beiden Strings ungleiche Längen und hat der Vergleich bis zum Ende des kürzeren Strings keine Unterschiede zwischen beiden Strings aufgewiesen, so ist beim Vergleich auf "größer" der längere String auch der größere, bzw. beim Vergleich auf "kleiner" der kürzere String der kleinere.

Beim Vergleich von Textausdrücken ist allerdings darauf zu achten, daß alle "normalen" Textzeichen einmal in großgeschriebener Form (A-Z/ASCII 65-90) und einmal kleingeschrieben (a-z/ASCII 97-122) existieren. Um also einen tatsächlich gültigen Vergleich auf alphabetische Reihenfolge durchzuführen, müßten beide Textausdrücke vorher vollständig entweder in Groß- oder Kleinschrift übertragen werden (siehe UPPER$).

```
PRINT "ABC"<"ABCD"="abc">"abcd"
```

Dieses fiktive Beispiel ist sicher nicht auf Anhieb zu verstehen. Zuerst werden die ersten beiden Ausdrücke getestet. Da ABC kleiner ist als ABCD, ergibt sich daraus der Wahrheitswert -1. Anschließend ergibt sich aus dem Vergleich abc > abcd der Wahrheitswert 0. Diese beiden Wahrheitswerte werden nun auf Gleichheit getestet. Da sie ungleich sind, wird der Wert 0 (= falsch) geliefert.

Enthalten die Strings auch andere Zeichen als die von A-Z (Ziffern, Leerzeichen, Satzzeichen), werden diese nach demselben Schema in den Vergleich mit einbezogeh. Siehe dazu auch unter MAX bzw. MIN.

4.16 Vorfahrtsregeln

Richtig heißt es natürlich Vorrangregeln oder Prioritäten. Gemeint ist damit die Reihenfolge, nach welcher die verschiedenen arithmetischem Operationen ausgeführt werden. Aus der Mathematik werden wohl den meisten Konstrukte wie das folgende oder ähnliche bekannt sein:

```
X=Sqr(12^3+(36-22^1.2)-(-4/3))*Sin(13)
```

Formeln dieser Art werden nach folgender Rangfolge aufgelöst:

1. Funktionen: Sqr, Tan, Atn etc. sowie DEFFN-Funktionen.
2. Klammern: ( ) Erst innere, dann äußere Klammern.
3. Potenzierung: ^
4. Negation: -
5. Multiplikation/Division: * , /
6. Modulo/Ganzahldivision: MOD , DIV od. \
7. Addition/Subtraktion: + , -

Da auch Vergleichsausdrücke und logische Verknüpfungen in dieser Form angegeben werden können, werden diese in die Rangfolge mit eingeordnet.

8. Vergleichsoperatoren: =, ==, <>, >, <, =>, <=
9. Logische Operatoren: AND,OR,NOT,XOR,IMP,EQV

```
PRINT (12^3+(36-22^1.2)-(-4/3))*3>14^3
```

oder

```
A=(12^3+(36-22^1.2)-(-4/3))*3 AND 14^3
```

Werden innerhalb eines arithmetischen Ausdrucks nur gleichwertige Operatoren verwendet, werden diese der Reihe nach von links nach rechts ausgeführt. Wollen Sie diese Reihenfolge durchbrechen, können Sie die Berechnungen, die zuerst behandelt werden sollen, in Klammern einfassen. In einer Klammer werden die Operationen wieder in der üblichen Rangfolge bearbeitet.

4.17 Fingerübungen

Im bisherigen Verlauf dieses Kapitels wurden schon einige BASIC-Befehle vorgestellt. Da ihre Verwendung sich aus dem jeweiligen Zusammenhang ergab, gehe ich davon aus, daß hier eine Vorstellung der wichtigsten Grundlagenbefehle und ihre Verwendungsmöglichkeiten angebracht ist.

Wie in Programmier-Lehrbüchern üblich, besteht das erste Programm eines Computerneulings daraus, den Satz "Hello World" auf dem Bildschirm auszugeben. Ich möchte hier nicht nachstehen, wobei ich der Meinung bin, daß in BASIC die Bildschirmausgabe eines beliebigen Textes derart leicht ist, daß ein erheblicher Lernerfolg daraus nicht herzuleiten ist.

Die Initialisierung und Bedienung des Computers wird Ihnen wohl dank der Atari-Bedienungsanleitung mittlerweile geläufig sein. Und wie der GFA-Interpreter gestartet wird, bzw. welche Bedienungsfunktionen zu beachten sind, wurde Ihnen ja bereits weiter vorn erläutert. Sie haben nun - bis auf die zwei Menüzeilen am oberen Bildrand - einen leeren Editor-Bildschirm vor sich. Der Cursor - das ist das kleine schwarze Rechteck oben links in der Ecke - zeigt Ihnen an, wo bei der nächsten Tastenbedienung ein Schriftzeichen erscheinen wird.

Geben Sie nun bitte über die Tastatur ein:

```
p "Hello World
```

und drücken Sie anschließend die <Return>-Taste. Hier begegnet Ihnen gleich ein wesentlicher Vorteil des GFA-BASICs. Die meisten Befehle können anhand von Kürzeln eingegeben werden. Sie werden bemerkt haben, daß der Interpreter selbstständig die Eingabe auf

```
Print "Hello World"
```

erweitert hat. Dieser Komfort hat bei einigen Befehlen erhebliche Wirkung, da es mit Sicherheit wesentlich schneller geht, z.B. statt GRAPHMODE einfach nur ein "g" schreiben zu müssen.

Probieren Sie es aus: Schreiben Sie einfach in die nächste Zeile ein "g 1" (in V3.0 "gr 1"), drücken Sie die <Return>-Taste und schon steht da Graphmode 1. Fahren Sie nun mit der Cursor-Taste Pfeil aufwärts (die Taste zwischen Insert und ClrHome im mittleren kleinen Tastenblock) den Cursor eine Zeile aufwärts auf die Graphmode-Zeile, drücken Sie <Control>+<Delete> gleichzeitig und schon ist die zweite Zeile auf Nimmerwiedersehen verschwunden.

Außerdem müssen bei Texteingaben nur die Einführungszeichen gesetzt werden. Das BASIC erkennt das Textende selbständig.

```
PRINT "Text
```

oder

```
A$="Text
```

Wir haben es bei dem GFA-BASIC-Interpreter mit einem äußerst komplexen Programm zu tun, dessen Leistungen erst bei längerer Arbeit deutlich werden.

Ich vergleiche es für mich immer mit einem Qualitätsauto. Der Unterschied zwischen drei bestimmten deutschen Autofirmen und ihren Konkurrenten besteht meistens in den fast unmerklichen Kleinigkeiten. Und wenn es nur das Geräusch des Türöffnens ist. Bei der einen Marke tut man es, weil man einsteigen will. Bei der anderen macht man die Tür wieder zu, um sie noch einmal öffnen zu dürfen - das Geräusch ist so schön! Dieses Beispiel ist natürlich stark übertrieben, macht jedoch das Wesen der Qualität deutlich: Auf die Kleinigkeiten kommt es an. Sie werden bei der Arbeit mit GFA-BASIC immer wieder auf solche angenehmen Kleinigkeiten stoßen und es sollte Ihnen selbst bei der Entwicklung Ihrer eigenen Programme immer naheliegen, auf diese Kleinigkeiten zu achten.

Aber weiter mit unserem ersten Programm. Sie haben nun mehrere Möglichkeiten, Ihr kleines Programm zu starten.

1. Sie können mit dem Mauszeiger nach rechts oben in die Bildschirmecke auf "Run" fahren und die linke Maustaste drücken.

2. Sie können <Shift>-Taste und <F10> gleichzeitig drücken.

3. Sie schalten in den Direktmodus um und geben dort den Befehl "ru" (Abk. für "Run") ein, gefolgt von der <Return>-Taste.

In jedem dieser Fälle wird der Bildschirm gelöscht, Ihr freundlicher Gruß oben links auf den Bildschirm geschrieben und eine sogenannte Alert-Box mit dem Hinweis "Programmende" ausgegeben. Nachdem Sie nun die <Return>-Taste gedrückt oder mit dem Mauszeiger auf "Return" geklickt haben, kehrt der Interpreter wieder zum Eingabe-Editor zurück. Damit dürfte eigentlich der Grundstein für Ihre Programmiererkarriere gelegt sein. Alles andere ist nur Übung.

Das Spiel, das wir oben veranstaltet haben, ist im eingentlichen Sinne noch nicht als Programm zu bezeichnen. Bei einer einfachen PRINT-Anweisung handelt es sich lediglich um einen einzelnen Programmschritt. Erst das sinnvolle Aneinanderreihen von mehreren Programmschritten bildet ein Programm.

Für den Anfang kann es jedoch nützlich sein, sich nach und nach mit den Möglichkeiten vertraut zu machen. Um ein wenig zu probieren, klicken Sie mit dem Mauszeiger im Menükopf des Editors das Feld "Direct" an. Sie sehen jetzt am linke Bildrand die Eingabeaufforderung "OK >". Dies ist der sogenannte Direkt- oder Kommandomodus. Auf dieser Ebene haben Sie nicht die Möglichkeit Programme zu schreiben, sondern können jeweils nur eine Befehlszeile (max. 255 Zeichen) eingeben, die nach Druck auf die <Return>-Taste direkt ausgeführt wird.

Geben Sie bitte ein:

```
OK >p "Hello World
```

Gleich unter der Eingabezeile erscheint nun:

```
Hello World
```

Sie sehen, daß jeder Befehl sofort nach <Return> ausgeführt wird. Im Direktmodus haben Sie also eine gute Möglichkeit, erst einmal einige Befehle in Ruhe auszuprobieren. Es gibt zwar Befehle, die hier nicht

ausführbar sind (Befehle, die mehr als eine Zeile für ihre korrekte Struktur benötigen), aber zur Übung ist der Direktmodus bestens geeignet.

Versuchen Sie es noch einmal:

```
OK >a 1,"Meine erste|Alert-Box",1," Na |toll",a
```

und nun <Return> drücken. So einfach ist das! Wenn Sie genug geübt haben, gelangen Sie durch <Esc> und anschließendem <Return> oder durch Drücken der Abbruchtasten <Control/Shift/Alternate> wieder zurück in den Editor.

Wieder im Editor? Gut, dann schreiben Sie nun Ihr erstes richtiges Programm.

Das wichtigste an einem Programm sind in jedem Fall die Eingabe- und Ausgabe-Anweisungen. Sicher sind andere Befehle auch wichtig, aber ohne die E/A-Befehle (in Engl.: I/O = Input/Output) geht gar nichts. Also werden wir damit beginnen:

```
REPEAT
  CLS
  INPUT "Wie heissen Sie ";Name$
  INPUT "Wie alt sind Sie ";Alter%
  PRINT "Drücken Sie bitte <Return>"
UNTIL INP(2)=13
PRINT "Ihr Name ist also ";Name$
PRINT "Sie sind ";Alter%;" Jahre alt."
```

Innerhalb der REPEAT...UNTIL-Schleife wird Ihr Name und Alter erfragt. Zuvor wird durch CLS der Bildschirm gelöscht und der Cursor in die linke, obere Bildschirmecke gebracht. Im Anschluß an die Eingaben, die jeweils durch <Return> abzuschliessen sind, werden die Daten in Variablen gespeichert. Der Name als STRING in einer String-Variablen und das Alter als numerischer Wert in einer Integervariablen. Danach werden Sie durch den PRINT-Befehl aufgefordert, nochmals <Return> zu drücken. In die UNTIL-Bedingungsabfrage wurde ein zweiter Eingabe-Befehl eingebunden, der auf eine einzelne Taste wartet. Wird nun <Return> (ASCII-Wert = 13) gedrückt, wird die Schleife verlassen und Ihre Eingabe - mit einem Kommentar versehen - wieder ausgegeben. Drücken Sie eine andere Taste, wird die Schleife wiederholt, indem zuerst der Bildschirm wieder gelöscht wird und das Spiel von vorn beginnt.

An den beiden abschließenden PRINT-Befehlen ist zu erkennen, daß die Ausgabe varriert werden kann. Durch das Format-Symbol ';' können PRINT- Ausgaben verkettet werden. Dabei ist es egal, in welcher Reihenfolge die Ausdrücke stehen.

Eine PRINT-Zeile kann z.B. auch so aussehen:

```
A$="Text"
PRINT LEN(A$)''''A$;" in GFA-";CHR$(66);"ASIC",CRSCOL'INP(2);
```

Eine solche Zeile kann also sehr bunt gemischt sein. Oben wird zuerst die Variable A$ mit dem Text Text initialisiert. Mit LEN(A$) wird die Länge des in A$ gespeicherten Textes, der Inhalt von A$ selbst, ein direkter String-Ausdruck (in GFA-), ein einzelnes ASCII-Zeichen (ASCII 66 = B), wieder ein Direkt-String (ASIC), die aktuelle Cursor-Position (CRSCOL) und zu guter Letzt eine Tastaturabfrage (INP(2)) in die Zeile eingebunden. Während(!) der Textausgabe werden die integrierten Funktionen bearbeitet und nach Erledigung ihr Ergebnis ebenfalls ausgegeben. So wird die Tastaturabfrage INP(2) erst bearbeitet, nachdem schon der übrige Text angezeigt wurde. Es sind generell alle Funktionen (ergebnisliefernde Operationen) in einer PRINT-Zeile einsetzbar.

Zur Verkettung der Ausdrücke wurden außerdem noch zwei weitere Format- Symbole verwendet. Ein Komma zwischen den Ausdrücken bewirkt, daß die darauffolgende Ausgabe an der nächsten von 5 Tabulatorpositionen erfolgt (siehe PRINT). Ein Apostroph dagegen gibt nur ein einzelnes Leerzeichen an der Position aus, an der es im Ausdruck steht. Das Semikolon haben Sie oben schon kennengelernt. In diesem Fall wurde es ganz am Ende der Zeile eingesetzt, um zu erreichen, daß die nächste PRINT-Ausgabe (bzw. auch OUT und WRITE) oder INPUT-Eingabe (oder auch INP, INPUT$ etc.) direkt hinter dem zuletzt angezeigten Text positioniert wird.

Durch

```
PRINT AT(10,10);
INPUT "Eingabe: ",A$
```

können Sie also auch die Position von Eingabe-Anweisungen bestimmen, wobei jedoch nach erledigter Eingabe der Cursor generell an den Anfang der nächsten Zeile zurückspringt. Im letzten Beispiel habe ich Ihnen wieder zwei neue Varianten vorgestellt. Durch den Zusatz AT(X,Y) kann die Cursor-Position beliebig auf dem Bildschirm bestimmt werden (siehe PRINT). Außerdem habe ich in der INPUT-

Zeile ein Komma hinter dem Eingabekommentar eingesetzt, um damit zu erreichen, daß die Eingabe ohne Frage- und Leerzeichen direkt hinter dem Kommentar beginnt.

Vergleichen Sie:

```
INPUT "Eingabe: ";A$
INPUT "Eingabe: ",A$
```

Unter den Befehlsbeschreibungen finden Sie noch weitere E/A-Befehle, die an Möglichkeiten fast keine Wünsche mehr offen lassen.

Versuchen wir uns nun ein wenig an der Grafik.

Möchten Sie lieber Kreise oder Rechtecke zeichnen? Oder vielleicht eine Freihandlinie? Alles kein Problem! Fortschrittliche BASIC-Dialekte haben selbstverständlich ein reichhaltiges Angebot an grafischen Befehlen. Das ist ganz und gar nicht so selbstverständlich, wie es uns heutzutage erscheint. Vor nur wenigen Jahren mußten die Algorithmen dazu noch per Hand geschrieben werden - und wer weiß schon aus dem Stehgreif, wie ein Kreis oder eine Ellipse konstruiert wird.

Wir haben es da einfach. In der Befehlsliste im Anhang finden Sie eine Fülle von Grafikbefehlen, deren Bedienung denkbar einfach ist. Unser Bildschirm ist in Hires (High Resolution = engl.: hohe Auflösung), also bei Betrieb eines Schwarz/Weiß-Monitors in 640 Bildpunkte horizontal und 400 Bildpunkte vertikal eingeteilt. Der Bildpunkt mit der Koordinate 0/0 liegt üblicherweise (siehe OPENW) in der äußersten linken, oberen Bildschirmecke. Durch dieses Wissen läßt sich nun jeder einzelne Bildschirmpunkt auch einzeln benennen. Sie müssen sich also nur entscheiden, an welcher Bildschirmposition Sie ein Grafik-Objekt sehen möchten, übergeben dem Befehl die entsprechenden Koordinaten und schon erscheint das Objekt. Die folgende Zeile zeichnet eine Ellipse, deren Zentrum genau in der Mitte des Bildschirms liegt:

```
ELLIPSE 320,200,310,190
```

Die ersten beiden Parameter bestimmen die Position, der dritte den horizontalen Radius und der vierte den vertikalen Radius der Ellipse. Diese Koordinatenangaben beziehen sich - wie gesagt - auf Hires. Bei Farbmonitoren gibt es zwei Auflösungsstufen. Die eine ist Midres (Middle Resolution = engl.: mittlere Auflösung) mit einer Einteilung von 640 horizontalen und 200 vertikalen Bildpunkten. Die zweite ist Lowres (Low Resolution = engl.: niedrige Auflösung) mit 320 hori-

zontalen und 200 vertikalen Punkten. Mit diesen Informationen und den Beschreibungen der Befehle werden Sie sicherlich keine größeren Probleme mit der Grafik haben.

Malen Sie doch mal ein kleines Bild mit dem folgenden Programm.

```
Deffill ,2,4              ! Füllmuster einstellen
Do                        ! Hauptschleifenstart --------.
  Repeat                  ! 1. REPEAT-Schleife -------. |
  Until Mousek            ! wartet auf Mausklick -----' |
  Mouse X,Y,K             ! Mauskoordinaten in X,Y      |
  Repeat                  ! 2. REPEAT-Schleife -------. |
    Mouse Xx,Yy,K         ! Neue Koordinaten in Xx,Yy | |
    If Mousek=1           ! Linke Maustaste?          | |
      Line X,Y,Xx,Yy      ! Dann Linie zeichnen       | |
      X=Xx                ! Neue Koordinaten werden   | |
      Y=Yy                ! Alte Koordinaten          | |
    Endif                 !                           | |
    If Mousek=2           ! Rechte Maustaste?         | |
      Flag!=True          ! merken                    | |
      Graphmode 3         ! Grafikmodus XOR           | |
      Box X,Y,Xx,Yy       ! Box zeichnen              | |
      Box X,Y,Xx,Yy       ! Box löschen               | |
      Graphmode 1         ! Grafikmodus Replace       | |
    Endif                 !                           | |
    If Mousek=3           ! Beide Maustasten?         | |
      Flag!=False         ! Merker löschen            | |
      Fill Xx,Yy          ! Ab Mausposition füllen    | |
    Endif                 !                           | |
  Until Mousek=0          ! Maustaste losgelassen?  --' |
  If Flag!=True           ! Merker für Maustaste 2?     |
    Flag!=False           ! Merker löschen              |
    Box X,Y,Xx,Yy         ! Box nochmal zeichnen        |
  Endif                   !                             |
Loop                      ! Zurück zum Anfang ----------'
```

Hier finden Sie gleich ein Beispiel für den Einsatz von Flags. Flag! ist hier eine völlig beliebige Boole-Variable, die die Aufgabe hat, sich zu merken, ob in der 2. REPEAT-Schleife die rechte Maustaste gedrückt wurde. Wurde sie das, wird im XOR-Modus zweimal eine Box auf dieselbe Stelle gezeichnet. Dieses hat den Effekt, daß die Box bei jedem Durchlauf nur für einen kurzen Moment zu sehen ist. Wird die rechte Maustaste wieder losgelassen, wird anschließend die Box noch einmal im REPLACE-Modus (siehe GRAPHMODE) gezeichnet.

Achten Sie bitte darauf, daß Sie, wenn Sie den Füllprozeß mit beiden Maustasten gleichzeitig auslösen wollen, zuerst die rechte und dann die linke Taste drücken, da sonst im Linienmodus schon ein Punkt gezeichnet wird, ehe der Füllprozeß beginnt. Dazu eine kleine Spielerei:

```
Plot 320,200                 ! Linienstart
For I=0 To 220 Step 0.1      ! 10tel-Step-Schleife ----.
  Add A,0.1                  ! Radius vergrößern       |
  X%=320+Cos(I)*A*1.6        ! X-Koordinate            |
  Y%=200+Sin(I)*A            ! Y-Koordinate            |
  Draw  To X%,Y%             ! Linie zeichnen          |
Next I                       ! Nächster Schritt -------'
```

Beachten Sie hier, daß die FOR...NEXT-Schleife mit einer Realvariablen (I) läuft, da Integervarialen nur Ganzzahlen aufnehmen können. Mit einer Integer-Zählvariablen würde das Programm lediglich eine gerade Linie zeichnen und unendlich weiterlaufen, da die Zählvariable immer wieder auf Null gesetzt würde. Die Linie ergibt sich dann ausschließlich aus der Erhöhung des Radius durch den Real-Faktor "A".

Das folgende Beispiel könnte für die absoluten Computer-"Greenhorns" unter Ihnen auf den ersten Blick etwas verwirrend wirken. Es stellt eine weitere Möglichkeit dar, mit Grafik zu jonglieren. Halten Sie bitte die Grafik-Programmierung nicht für nebensächlich! Manche Probleme lassen sich durch einen geschickten Einsatz von Grafik wesentlich vereinfachen. Es handelt sich also nicht nur um schmückendes Beiwerk. Ein Bild sagt mehr als tausend Worte. Sollten Ihnen die folgenden Erläuterungen nicht ausreichen, schlagen Sie bitte bei den entsprechenden Befehlen nach.

```
' ** Zeichnen der Linienzüge **
Restore GFA                      ! DATA-Zeiger auf GFA-Label setzen
Repeat                           ! Äußere Schleife >-------------.
  Read X%,Y%                     ! Linien-Startkoordinaten lesen  |
  Do                             ! Innere Schleife >------------.|
    Read Xx%,Yy%                 ! Linien-Endkoordinaten lesen  ||
    Exit if Xx%<0                ! Abbruch bei Endmark. -1 oder -2||
    Line X%,Y%,Xx%,Yy%           ! Linie zeichnen               ||
    X%=Xx%                       ! Endkoordinate = Startkoordinate||
    Y%=Yy%                       !              der nächsten Linie||
  Loop                           ! Schleifen-Wendepunkt <--------'|
Until Xx%=-1                     ! Abbruch bei Endmarkierung -1 <--'
' ** Versatz der Grafik und Füllen **
For J%=1 To 3                    ! 3mal   >---------------------.
  Read Xl%,Xr%                   ! Koordinaten lesen             |
  Get Xl%,100,Xr%,270,A$         ! Buchstaben speichern          |
  For I%=Xl% To Xl%+8 Step 2     ! 4mal                          |
    Put I%,100-(Xl%-I%),A$,7     ! Grafik transparent versetzen  |
  Next I%                                                        |
  Fill Xl%+9,190                 ! Buchstaben füllen             |
Next J%                          ! <-----------------------------'
Do                               ! Endlos-Schleife >-------------.
  Sget A$                        ! Full-Screen speichern          |
  For J%=0 To 19                 ! 20 mal >--------------------. |
    For I%=100+J% To 300+J% Step 20  ! 200 Pixel in 20er Schritten ||
      Bmove Varptr(A$)+I%*80,Xbios(2)+32000-I%*80,80 ! verdrehen  ||
    Next I%                                                     ||
  Next J%                        ! <---------------------------'|
```

```
Loop                                     ! <-----------------------------'
GFA:
Data 124,100,200,100,200,120,130,120,120,130,120,240
Data 130,250,190,250,200,240,200,180,150,180,150,160
Data 220,160,220,246,196,270,124,270,100,246,100,124
Data 124,100,-2,-2
Data 250,100,350,100,350,120,270,120,270,160,330,160
Data 330,180,270,180,270,270,250,270,250,100,-2,-2
Data 404,100,476,100,500,124,500,270,480,270,480,180
Data 400,180,400,270,380,270,380,124,404,100,-2,-2
Data 410,120,470,120,480,130,480,160,400,160,400,130
Data 410,120,-1,-1
Data 100,220,250,350,380,500
```

Soweit zur Einführung in die Programmiersprache GFA-BASIC und in die Hintergründe und Grundlagen der Computer-Programmierung. Es ist mir nicht immer leicht gefallen, die richtigen Worte zu finden, um ein Thema auch - und gerade - für den Computer-Anfänger verständlich und durchschaubar werden zu lassen. Ich habe es jedenfalls nach besten Kräften versucht und hoffe, daß Ihnen dieses Rüstzeug auf Ihrem weiteren Weg in der "hohen Schule" der Programmierkunst ein Stück helfen wird.

Sie finden zudem im Verlauf des Buches immer wieder kleine Einschübe, die dazu beitragen sollen, Ihre hier erworbenen Kenntnisse zu vertiefen. Noch ein kleiner Tip zum Schluß. Setzen Sie sich in der Anfangszeit so viel wie möglich mit fremden Programmen auseinander. Achten Sie dabei darauf, daß diese in ihrer Komplexität ungefähr Ihrem eigenen Wissensstand entsprechen und verändern Sie sie geringfügig.

5. Ein-/Ausgabebefehle

5.1 Dateneingabe

FORM INPUT { F MINPUT/V3.0: F } Formatierte String-Eingabe

FORM INPUT Anz,Var$

Eine Kombination aus INPUT- und INPUT$-Befehl ist der FORM INPUT-Befehl. Sie können damit eine maximale Länge des einzugebenden Textes vorherbestimmen. Die größtmögliche Länge des Strings liegt bei 255 Zeichen. Wird vom Anwender die vorgesehene String-Länge erreicht, bleibt der Cursor an der letzten Stelle stehen und es ertönt ein Alarmton (falls die Lautstärke an ist). Die Eingabe kann also nur durch <Return> abgeschlossen werden. Veränderungen am Text sind auf die gleiche Art und Weise möglich, wie unter INPUT beschrieben. Ebenso die Eingabe von Sonderzeichen. Die Angabe einer Variablenliste oder eines vorangestellten Textes ist hier allerdings nicht möglich. Es wird generell der Cursor auf die erste Stelle des Eingabebereichs gesetzt.

Beispiel:

```
FORM INPUT 32,Textvar$
```

Positionieren können Sie den Eingabebereich übrigens, wie auch bei INPUT und LINE INPUT, indem Sie mit

```
PRINT AT(XP,YP);
```

den Cursor an die Stelle bringen, wo das erste Zeichen der Eingabe erscheinen soll. Durch das dem PRINT-Befehl nachgestellte Semikolon wird der Wagenrücklauf (CR = Carriage Return) und der Zeilenvorschub (LF = Line Feed) unterdrückt. Das nächste Zeichen einer TOS-Ausgabe wird dann an die Position direkt hinter dem zuletzt ausgegebenen Zeichen gesetzt.

FORM INPUT AS

Formatierte String-Eingabe m. Vorgabe

{ F MINPUT AS/V3.0: F AS }

FORM INPUT Anz AS Var$

Hier gilt grundsätzlich das gleiche wie bei FORM INPUT. Nur daß hier der Inhalt einer String-Variablen auf dem Bildschirm ausgegeben und die nachträgliche Edition des darin enthaltenen Strings (oder eines Teils davon) ermöglicht wird.

"Anz" enthält die Anzahl der Zeichen ab Anfang des Vorgabe-Strings, die zur Edition ausgegeben werden sollen. Nach erneuter Eingabe und Bestätigung durch <Return> wird der neu entstandene String in die Variable übernommen. Der vorherige Variableninhalt wird **komplett** ersetzt. Wird mit 'Anz' die Länge der angegebenen Variablen nicht überschritten, muß vor der Edition anhand der <Delete>-Taste Platz zur Eingabe geschaffen werden. Bei Angabe einer Leervariablen erfüllt dieser Befehl dieselbe Funktion wie FORM INPUT.

Beispiel: (in Verbindung mit MID$()= und MID$)

```
A$="String vor der Eingabe!"           ! Beliebiger String
B$=Mid$(A$,7,4)                        ! 3 Zeichen austrennen
Print At(10,10);"=> ";A$;" => ";       ! Eingabe positionieren
Form Input 7 As B$                     ! 8 Zeichen zur Eingabe
Mid$(A$,8,Len(B$))=B$                  ! Eingabe einfügen
Print At(10,10);"=> ";A$;Spc(15)       ! String ausgeben
Void Inp(2)                            ! Auf Taste warten
```

INKEY$

Einzelzeichen von Tastatur holen

Zeichen$=INKEY$ => Zeichen-Zuweisung
[IF/WHILE/UNTIL] LEN(INKEY$) => Bedingung: Inkey$ >""
[IF/WHILE/UNTIL] INKEY$="Z" => Bedingung: "Z" gedrückt

Mit INKEY$ können nicht nur die normalen Tastenbelegungen erfragt werden, sondern auch alle Sondertasten (z.B. F1 - F10 und Cursor-Block). Zusätzlich werden fast alle Kombinationen von <Control>, <Shift> oder <Alternate> mit anderen Tasten registriert. So lassen sich die Funktionstasten in Kombination mit der Shift-Taste doppelt belegen. Die Funktion schaltet intern auf einen anderen Modus um, sobald Sondertasten bzw. ihnen entsprechende Tastenkombinationen gedrückt werden. Im Normalfall wird ein Ein-Zeichen-String zurückgegeben, der das der gedrückten Taste entsprechende ASCII-Zeichen beinhaltet. Bei Sondertasten wird dagegen ein Zwei-Byte-String zurückgegeben,

der im ersten Byte eine Null enthält und im zweiten Byte dann einen Code, der zwar ebenfalls ASCII-Zeichen produziert, aber eigentlich nichts mehr mit diesen zu tun hat. Dieser Code leitet sich aus einer systeminternen Tasten-Identifikations-Matrix ab, die Scan-Code genannt wird. Damit lassen sich dann (fast) sämtliche Tasten, die auf der Tastatur zu finden sind und (fast) alle Tastenkombinationen gezielt identifizieren. Wird im Moment der INKEY$-Abfrage keine Taste gedrückt, wird ein Leer-String ("") geliefert.

In der V3.0-Version sind hier allerdings die durch KEYPAD eingestellten Tastaturattribute zu beachten. Außerdem haben Sie in V3.0 eine erweiterte Auswahl an Tastatur-Abfragen, die -je nach Einsatzgebiet - zum Teil noch um eine Kleinigkeit komfortabler sind (siehe KEYxxx-Befehle).

Beispiel:

```
Print "Testen Sie beliebige Tasten und Tastenkombinationen"
Print "              (Abbruch durch <Esc>)
Do                                    ! Große Schleife
  Repeat                              ! Eingabeschleife
    Key$=Inkey$                       ! Abfrage
  Until Key$>""                       ! bis Taste gedrückt ist
  Print At(10,10);                    ! Ausgabe positionieren
  If Len(Key$)=1                      ! Ein-Byte-Code
    Print "ASCII :  ";
    Out 5,Asc(Key$)                   ! TOS-Direkt-Ausgabe
    Print ''Asc(Key$)'''''''          ! ASCII-Code
  Else                                ! Zwei-Byte-Code
    Print "Scan :   ";
    Out 5,Asc(Right$(Key$))           ! TOS-Direkt
    Print ''Asc(Right$(Key$))'''''! Scan-Code
  Endif
  Exit If Key$=Chr$(27)               ! Abbruch bei <Esc>
Loop                                  ! Schleifenende
```

Da gibt es allerdings ein Problem. Der Tastatur-Puffer hat die - manchmal unangenehme - Eigenart, sich die Tasten, die gedrückt wurden, zu merken. Und zwar alle! Das führt dazu, daß - wenn man ganz kontrolliert nur einen Tastendruck zulassen will - über INKEY$ alle Zeichen ausgegeben werden, die im Tastaturspeicher evtl. durch einen Dauertastendruck gespeichert wurden. Dieses Problem läßt sich beseitigen, indem man nach der INKEY$-Abfrage eine kleine Schleife einbaut, die den Tastaturpuffer leert.

```
Repeat
Until Inkey$=""
```

Derselbe Effekt läßt sich erzielen, wenn man in den Orga-Bereich des Tastaturpuffers an die Stelle, wo die momentan vorhandene Zeichenanzahl des Puffers gespeichert wird, einfach eine Null LPOKEd.

```
Lpoke Xbios(14,1)+6,0
' {Xbios(14,1)+6}=0  ! (V3.0-Variante)
```

Um eine Einzeltastenabfrage durch INKEY$ zu realisieren (z.B. in einem alphanumerisch indizierten Menü) kennt man aus anderen BASICs die Variante

```
100 A$=Inkey$:If A$="" Then Goto 100
```

Diese Konstruktion wird in GFA-BASIC ersetzt durch:

```
Repeat
Until Len(Inkey$)
```

oder

```
Repeat
Until Inkey$<>""
```

Sie hat die gleiche Aufgabe, nämlich darauf zu warten, daß irgendeine Taste auf dem Keyboard gedrückt wird.

Um eine bestimmte Taste zu erfragen, gibt es in anderen BASIC-Dialekten die altbekannte Möglichkeit:

```
100 A$=Inkey$:If A$<>"b" Then Goto 100
```

Das gleiche in GFA-BASIC:

```
Repeat            Until Inkey$="b"
```

Damit kann also kontolliert werden, ob eine bestimmte Taste (hier: b) gedrückt wurde. Wird eine andere Taste gedrückt, bleibt das Programm in der Warteschleife.

Manchmal kann es unter bestimmten Umständen zu Verwechslungen zwischen ASCII- und Pseudo-Scan-Code kommen, da eben auch die Scan-Codes innerhalb des normalen alphanumerischen Bereichs (a - z, A - Z, 0 - 9 und Interpunktion) liegen können und außerdem aus dem Code nicht unbedingt zu erkennen ist, mit welcher Umschalttaste (<Control>, <Shift>, <Alternate>) nun der Zwei-Byte-Code produziert wurde.

Wenn man ganz sicher gehen will, empfiehlt es sich, zusätzlich zur normalen INKEY$-Analyse (Key%=ASC(RIGHT$(INKEY$))) noch gezielt den Status der Umschalttasten zu ermitteln. Dies erreichen Sie durch die BIOS-Funktion 11:

```
Skey%=BIOS(11,-1)
```

Hier erhalten Sie in Skey% (beliebiger Variablenname) einen 5BIT-Wert mit folgender Bedeutung:

```
BIT 0 an (IF Skey% AND 1) = Rechte Shift-Taste gedrückt
BIT 1 an (IF Skey% AND 2) = Linke Shift-Taste gedrückt
BIT 2 an (IF Skey% AND 4) = Control-Taste gedrückt
BIT 3 an (IF Skey% AND 8) = Alternate-Taste gedrückt
BIT 4 an (IF Skey% AND 16)= CapsLock ist angeschaltet
```

Der Wert 22 würde also z.B. aussagen, daß zum Zeitpunkt der Abfrage CapsLock angeschaltet war und gleichzeitig <Shift-links> und <Control> gedrückt wurden.

Wollen Sie eine Warteschleife realisieren, die auf die Betätigung einer bestimmten Umschalttaste (hier <Alternate>) wartet, gehen Sie wie folgt vor:

```
Repeat
Until Bios(11,-1) And 8
```

Mit dieser BIOS-Funktion haben Sie auch die Möglichkeit, eine bestimmte Umschalttaste oder Umschalttastenkombination zu simulieren, bzw. den Status der Umschalttasten auf Null zu setzen. Wie Sie bereits erfahren haben, gibt es in GFA-BASIC eine Programm-Abbruchfunktion, die durch gleichzeitiges Drücken von <Control>, <Shift> und <Alternate> ausgelöst wird. Über ON BREAK GOSUB Prozedurname läßt sich diese Funktion unterbinden und statt dessen zu einer beliebigen Prozedur verzweigen.

```
On Break Gosub Abbruch        ! Break-Versuch umleiten
Do
  Key%=Asc(Right$(Inkey$))    ! Tastatur-Abfrage
  If Key%>0                   ! Taste gedrückt?
    If Key%<>27               ! <Esc> nicht gedrückt?
      Void Bios(11,14)        ! Dann Break-Funktion simulieren
    Else                      ! <Esc> gedrückt !
      Edit                    ! Dann Programmende
    Endif
  Endif
Loop
Procedure Abbruch
  Void Bios(11,0)             ! Status auf Null setzen (Break Off)
```

```
  Print "Abbruch nur durch <Esc>"
Return
```

So läßt sich z.B. auch CapsLock beliebig ein- und ausschalten:

```
Void Bios(11,16)            ! CapsLock einschalten
  Input "beliebige Eingabe (<Return>=Weiter) :";A$
  Void Bios(11,0)             ! CapsLock ausschalten
  Input "beliebige Eingabe (<Return>=Weiter) :";A$
```

Möchten Sie eine 100%ige Identifikation erreichen, haben Sie noch eine weitere Möglichkeit, die durch eine GEMDOS-Funktion alles erreicht:

```
Deffn Scan=Gemdos(6,255)
(als Funktion definiert; Aufruf: @Scan)
```

Die Arbeitsweise dieser Funktion ist identisch mit INKEY$. Es wird nicht auf einen Tastendruck gewartet, sondern der zum Zeitpunkt des Aufrufs auf der Tastatur erzeugte Code oder Null zurückgegeben. Sie erhalten:

1. In den unteren 8 Bit den ASCII-Code der Taste:

```
Ascii%=@Scan And 255
```

2. In den Bits 16-23 den echten Scan-Code der Taste:

```
Scan%=@Scan/2^16 And 255
```

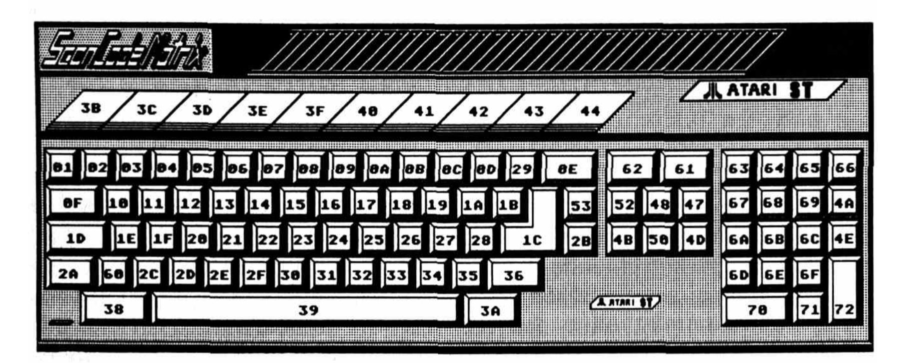
3. In den Bits 24-31 den Status der Umschalttasten (nur gleichzeitig mit anderer Taste oder wenn CapsLock an):

```
Status%=@Scan/2^24 And 255
```

Beispiel:

```
Do
  Key%=@Scan                  ! Tastatur-Code holen
  Ascii%=Key% And 255         ! ASCII-Code
  ' Ascii%=And(Key%,255)            ! (V3.0-Variante)
  Scan%=Key%/2^16 And 255     ! Scan-Code
  ' Scan%=And(Swap(Key%),255)       ! (V3.0-Variante)
  Status%=Key%/2^24 And 255 ! Umschalttasten-Status
  ' Status%=And(Swap(Key%),&HFF00) ! (V3.0-Variante)
  If Key%                     ! Taste gedrückt?
    Print At(10,10);Key%'',Ascii%'',Scan%'',Status%''
  Endif
Loop
Deffn Scan=Gemdos(6,255)     ! Funktionsdefinition
```

Durch den in Key% gelieferten Gesamtwert wird **jede** Taste und **jede** mögliche Tastenkombination eindeutig identifizierbar. Damit dürfte die Tastenanalyse kein Problem mehr für Sie darstellen.



INPUT { INP } — Dateneingabe

INPUT ["Text";,] Var1 [,Var2,...]
INPUT #Kanal,Var1 [,Var2,...]

Dieses ist der auf TOS-Ebene noch am meisten verwendete Befehl zur Eingabe von Werten oder Strings an das Programm. Im Anschluß an diesen Befehl kann eine Text-Konstante angegeben werden, die vor der ersten Eingabestelle erscheinen soll.

Nach einem Komma oder Semikolon wird dann die Werte- oder Textvariable angegeben, die die eingegebenen Daten aufzunehmen hat. Der Unterschied zwischen der Angabe eines Kommas oder eines Semikolons ist der, daß bei einem Semikolon nach dem Befehl, bzw. nach dem eingefügten Text ein Leer- und ein Fragezeichen erscheint, während beim Komma direkt nach dem Befehl oder Text der Cursor erscheint und dort die Eingabe beginnen kann.

Sollen mehrere Werte oder Strings über einen Input-Befehl eingegeben werden, können Sie durch Kommas getrennt eine Liste der dafür vorgesehenen Variablen anfügen. Der Befehl ist dann erst abgeschlossen, wenn für jede angegebene Variable die entsprechenden Daten eingegeben wurden. Dabei ist es sogar möglich, verschiedene Variablentypen zu verwenden. Sie können somit also mit einem Input-Befehl gleichzeitig Werte und Text erfragen. Weisen Sie jedoch dem angegebenen Variablentyp nicht die entsprechenden Daten zu (bei numerischen Variablen Text oder umgekehrt), wird ein akustisches Signal

ausgegeben (sofern Sie die Lautstärke Ihres Monitors nicht ausgeschaltet haben) und die Eingabe kann wiederholt werden.

Jede einzelne Eingabe kann bei Mehrfach-INPUT entweder durch <Return> abgeschlossen werden oder Sie können die Daten innerhalb einer Zeile jeweils durch ein Komma voneinander trennen. Bei Trennung durch <Return> wird dann allerdings der Cursor an den Anfang der nächsten Bildschirmzeile gesetzt und dort auf die nächste Eingabe gewartet.

Möchten Sie Kommas in der Eingabezeile verwenden, ohne daß dadurch zur nächsten Variablen weitergeschaltet wird, erreichen Sie dies, indem Sie die betreffende Antwort in Anführungsstriche setzen. Bei Verwendung einer einzelnen String-Aufnahmevariablen und Verwendung von Kommas, ohne daß die Antwort in Anführungsstriche gesetzt wurde, wird die Antwort bei dem ersten auftretenden Komma abgeschnitten und nur dieser vordere String-Teil der Variablen zugeordnet.

Beispiel:

```
Print At(10,10);
Input "Eingabe Wert,String,Wert: ",A%,B$,C%
Print A%,B$,C%
```

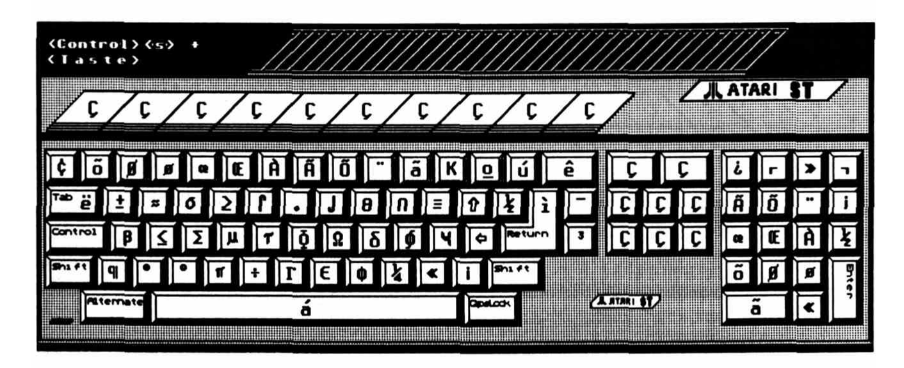
Bei String-Eingaben sind Strings mit einer Länge von bis zu 255 Zeichen möglich. Während der Eingabe können Sie die schon eingegebenen Daten auf verschiedene Weise korrigieren:

```
<Backspace>    = Zeichen links vom Cursor löschen
<Delete>       = Zeichen unter dem Cursor löschen
<Pfeil-links>  = Cursor um eine Stelle nach links
<Pfeil-rechts> = Cursor um eine Stelle nach rechts
                 (sofern er nicht am Zeilenende steht)
<Pfeil-hoch>   = Cursor zum Zeilenanfang
<Pfeil-runter> = Cursor zum Zeilenende
```

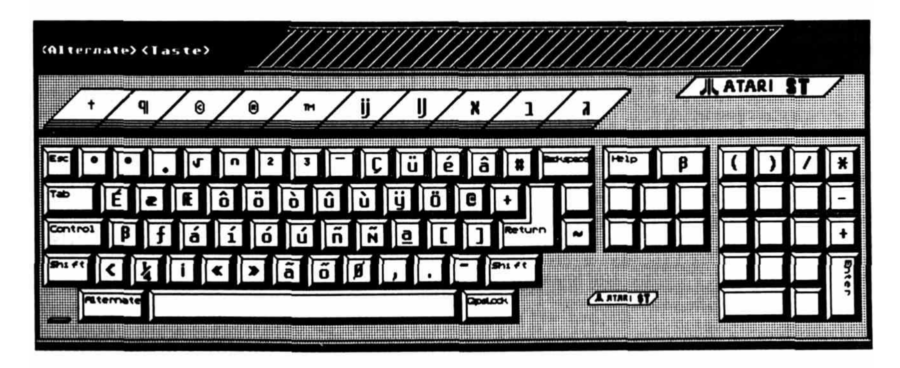
In V3.0 können Sie zusätzlich durch <Insert> zwischen Überschreib- und Einfügemodus wählen.

Wollen Sie bei der Eingabe Sonderzeichen verwenden, gibt es dazu drei Möglichkeiten:

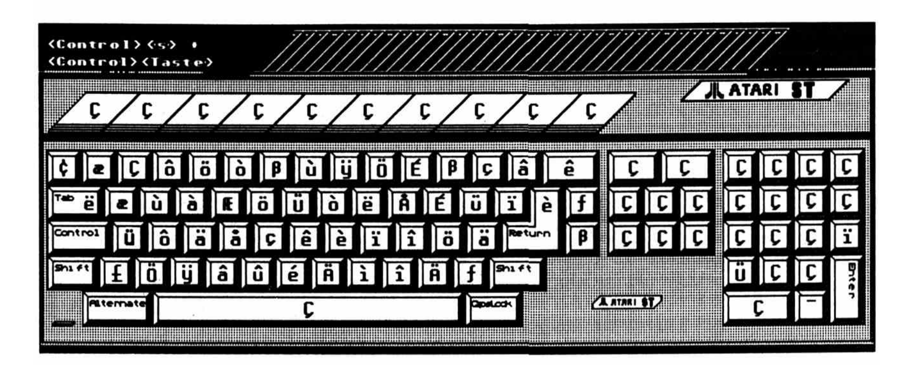
1. <Alternate> und andere <Taste>.



2. Gleichzeitig <Control> und <s> drücken, anschließend eine andere <Taste>.



Wird nach <Control> und <s> zusätzlich zur <Taste> nochmals <Control> gedrückt, ergibt sich eine andere Belegung.



3. Gleichzeitig <Control> und <a> drücken, anschließend den ASCII-Wert des gewünschten Zeichens eingeben. Zeichen mit einem ASCII-Wert zwischen 0 und 32 müssen nach Eingabe des Wertes mit <Return> quittiert werden.

   Diese lassen sich dann allerdings innerhalb der Eingabe nicht auf dem Bildschirm darstellen (Steuerzeichen).

Wie bei PRINT gibt es auch hier die Kanal-Variante. Mit INPUT # und der daran anschließenden Nummer des gewünschten Datenkanals können auch Daten aus Diskettendateien, bzw. über die Schnittstellen (siehe OPEN) bezogen werden.

Beispiel:

```
Open "I",#1,"DATEI.DAT"
Input #1,A$,B$,C$
```

oder:

```
Open "",#1,"AUX:"
Input #1,A$,B$,C$
```

Bei diesen Beispielen muß allerdings gewährleistet sein, daß die aus der Datei gelesenen Daten auch zu den vorgegebenen Variablentypen passen.

INPUT$() Zeichenketteneingabe

A$=INPUT$(Anz)
A$=INPUT$(Anz,#Kanal)

Wollen Sie, daß Strings "verdeckt" eingegeben werden, können Sie das mit dieser Eingabe-Funktion erreichen. Anders als bei allen anderen Input-Arten wird hier die aufnehmende Variable nicht dem Befehl nachgestellt, sondern der String wird einer Variablen zugewiesen (A$=INPUT$(10)), direkt nach Abschluß der Eingabe ausgegeben (PRINT INPUT$(10)) oder in eine Abfrage eingebunden werden (IF INPUT$(4)="xxxx").

Der Funktion wird in Klammern eine Zahl übergeben, die angibt, wieviele Zeichen maximal eingegeben werden können. Wird diese Zeichenanzahl erreicht, wird das Programm, ohne auf die Betätigung der <Return>-Taste zu warten, fortgesetzt. Der eingegebene Text ist während der Eingabe nicht zu sehen. Korrekturen an der Eingabe sind auf dieselbe Art wie bei INPUT möglich.

Auch hier gibt es die Kanal-Variante (A$=INPUT$(10,#1)). Dieser Befehl eignet sich besonders dazu, um eine bestimmte Datenmenge aus einer Diskettendatei zu lesen. Um zum Beispiel eine ganze Datei in einen String einzulesen (vorausgesetzt, sie ist nicht länger als die maximale String-Länge von 32767 Byte), können Sie folgendes Mini-Programm verwenden (siehe auch BGET# bzw. LINE INPUT#) :

```
Open "I",#1,"Datei.Dat"
A$=Input$(Min(32767,Lof(#1)),#1)
Close #1
Print A$
Void Inp(2)
```

Sie müssen hierbei die Länge der Datei nicht kennen, da sie durch LOF() ermittelt wird. Um die maximal mögliche String-Länge (32767 Zeichen) nicht zu überschreiten, wird die einzulesende Zeichenanzahl durch MIN() auf diese maximale Anzahl begrenzt.

LINE INPUT { LI INPUT/V3.0: LI } Zeichenketteneingabe

LINE INPUT ["Text";,] Var$ [Var2$,...]
LINE INPUT #Kanal, Var$ [Var2$,...]

Der LINE INPUT-Befehl ist ein direkter Nachkomme des normalen INPUT-Befehls. Sie unterscheiden sich dadurch, daß LINE INPUT

ausschließlich für Texteingabe zu verwenden ist, und daß bei Angabe einer Variablenliste die Eingaben alle einzeln mit <Return> abgeschlossen werden müssen. Während bei INPUT durch die Eingabe eines Kommas kenntlich gemacht wird, daß nun die nächste Variable in der Liste bedient wird, wird hier das Komma als Textzeichen innerhalb des eingegebenen Strings anerkannt. Genauso wie bei INPUT kann auch hier ein Text-String übergeben werden, der dann vor dem ersten einzugebenden Zeichen erscheint.

Beispiel:

```
Line Input "Bitte Text eingeben : ",Aa$,Bb$,Cc$
```

Durch Verwendung eines Kommas oder Semikolons zwischen dem Befehl (bzw. Text) und der Variablen (bzw. Variablenliste) kann auch hier bestimmt werden, ob ein Leerzeichen und ein Fragezeichen vor der ersten Eingabestelle erscheinen soll oder nicht. Editionsmöglichkeiten wie bei INPUT.

Die zweite Syntaxform liest die Daten aus der in Kanal angegebenen Datei oder Schnittstelle (siehe OPEN). Hier erklärt sich auch, warum dieser Befehlsname mit LINE beginnt. Befinden sich nämlich innerhalb des eingelesenen Textes keine CR's oder LF's, bzw. zwischen einem CR/LF und dem nächsten liegen mehr als 254 Zeichen, wird der Text nach maximal 254 Zeichen automatisch abgeschnitten und der betreffenden Variablen zugeordnet. Texteingaben mit mehr als 254 Zeichen (254+CR+LF = 256 = max. Zeilenlänge) sind nicht möglich.

Sind im Text Zeichen mit einem ASCII < 32 enthalten, die keine Zeilentrennzeichen (CR/LF) sind, sind Fehleingaben möglich.

Etwas verwirrend ist, daß in Fällen, in denen nur ein Carriage Return zur Zeilentrennung verwendet wurde, dies zwar insoweit erkannt wird, daß nur der vordere Teil des Textes bis zum ersten CR an die Variable übergeben wird, jedoch der File-Pointer um die gesamte Anzahl der gelesenen Zeichen weitergesetzt wird.

Folgerung: LINE INPUT# ist nur für ASCII-Text (ASCII > 31) - mit CR+LF als Zeilentrennung - korrekt einsetzbar.

Beispiel (Hires/Midres/Lowres):

```
Deffill ,2,4
Pbox 10,10,300,190
Fileselect "\*.*","SHOW_ASC.DAT",F.nam$ ! Beliebige ASCII-Datei
```

```
If Exist(f.nam$)                       ! Datei gefunden?
  @Showdat(F.nam$)
Endif
Procedure Showdat(Selec$)               ! als Befehlserweiterung
  Local Backscreen$,Sh$,I%,Tkey%,Back%,Curln%,Byte%,Xt%,Old%
  Xt%=2-Sgn(Xbios(4))                   ! X-Auflösungsteiler
  Sget Backscreen$                      ! Hintergrund sichern
    Open "I",#99,Selec$                 ! Dann öffnen
    Open "",#98,"VID:"                  ! OPEN Monitor
    Cls                                 ! Clearscreen
    Repeat
      If Lof(#99)<255                   ! Dateigröße < 255
        Sh$=Input$(Lof(#99),#99)        ! Ganze Datei laden
        For I%=0 To Len(Sh$)-1          ! Datei durchgehen
          Byte%=Peek(Varptr(Sh$)+I%)    ! Jedes Byte...
          '   Byte%=Byte{V:Sh$+I%}      ! (V3.0-Variante)
          If Byte%=13 Or Byte%=10       ! auf CR bzw. LF testen
            Print Chr$(Byte%);          ! Steuerzeichen ausgeben
          Else                          ! Kein CR oder LF !
            Out 5,Byte%                 ! Byte direkt ausgeben
          Endif
        Next I%                         ! Nächstes Byte
        Print Mkl$(&H1B441B44)          ! Cursor zwei Zeichen nach
        ' links und CR/LF (&H1B441B44 = 2 mal CHR$(27)+"D")
      Else                              ! Dateigröße > 256
        Repeat
          If Eof(#99)=0                 ! Dateiende?
            Old%=Loc(#99)               ! Alten File-Pointer merken
            Line Input #99,Sh$          ! und Zeile lesen
            Print #98,Sh$;              ! ... und ausgeben
            Print Mkl$(&H1B441B44)      ! Cursor zwei Zeichen nach
            '                           ! links und CR/LF.
            ' Diese Cursor-Korrektur ist nicht unbedingt notwendig.
            ' Sie dient dazu, die auf "VID:" mit ausgegebenen CR/LF
            ' zu überdecken und eine optische Zeilentrennung zu
            ' erreichen.
            Seek #99,Old%+Len(Sh$)+2    ! File-Pointer neu setzen
            '        = Alter Pointer+String-Länge+CR+LF
            ' Dies ist notwendig, um bei "Nicht-ASCII-Strings", die
            ' ggfs. keine korrekte Zeilentrennung beinhalten, den
            ' richtigen Anschluß wiederzufinden.
            Te%=Int(Len(Sh$)/(80/Xt%))+Sgn(Len(Sh$) Mod (80/Xt%))
            ' ...String-Teiler berechnen
            Add Curln%,Te%
            ' ...Zeilenzähler entsprechend erhöhen
          Endif
          Tkey%=Asc(Right$(Inkey$))     ! Tastatur abfragen
        Until Curln%>21 Or Eof(#99) Or Mousek=2 Or Tkey%=27
        '...wenn 22 Zeilen oder Dateiende oder rechte Maustaste
        '   oder <Esc>, dann nächste Seite
      Endif
      If Eof(#99)                       ! Dateiende?
        Print " **** Dateiende ****"''  ! Dann Hinweis
      Else
        Print Chr$(27);"p";" Taste  (<Esc> = Abbruch)";Chr$(27);"q"
        ! Aufforderung zum Weitermachen (invertierte Schrift).
      Endif
      Repeat                            ! auf irgendeine...
        Tkey%= Asc(Right$(Inkey$))      ! ... Taste ...
```

```
        Until Tkey% Or Mousek                  ! ... warten
        Print Mkl$(&H1B491B4B);                ! Cursor eine Zeile hoch
        ' und Zeile löschen. (&H1B491B4B = CHR$(27)+"I"+CHR$(27)+"K")
        Clr Curln%                             ! Zeilenzähler auf Null
      Until Eof(#99) Or Tkey%=27               ! Abbruch wenn Dateiende...
      '                                        ! oder <Esc> gedrückt
      Close #98                                ! Monitor schließen
      Close #99                                ! Datei schließen
      Sput Backscreen$                         ! Hintergrund restaurieren
      Pause 10                                 ! Kleine Klickpause
Return
```

5.2 Datenausgabe

PRINT { ? oder P } — Daten ausgeben

PRINT [AT(S,Z)][,']"Text"[[;,']Var[;,']Expr...;]
PRINT [#Kanal,[;]] "Text"[[;,']Var[;,']Expr...;]

Mit PRINT läßt sich fast alles auf den Bildschirm schreiben. Er ist einer der Tausendsassa-Befehle die man in BASIC kennt. Um seinen Variantenreichtum ausloten zu können, wird es Ihnen nicht erspart bleiben, ihn in verschiedenen Fällen bei der Arbeit zu beobachten. Probieren Sie ihn also fleißig aus.

Beispiel:

```
A%=1
A$="- Demo"
Print At(1,12);
Print Chr$(27);"p";" PRINT"'A$,A%,"Taste: ";
Print Chr$(Inp(2))'"!";Chr$(27);"q";At(1,13);
Print String$(40,Left$(A$));Chr$(Min(32,Inp(2)))
```

In diese bunt zusammengesetzten PRINT-Befehlen haben die einzelnen Komponenten folgende Bedeutung:

At(XP,YP)
Mit diesem wahlfreien Zusatz können Sie bestimmen, an welcher Cursor-Position die Ausgabe erfolgen soll. Der TOS-Bildschirm besteht, von der Home-Position aus gezählt:

```
In Hires/Midres: Aus 80 Spalten und 25 Zeilen
                 (XP= 1-80/YP= 1 - 25)
In Lowres:       Aus 40 Spalten und 25 Zeilen
                 (XP= 1-40/YP= 1 - 25)
```

Dasselbe wird ab der V2.02-Version auch durch LOCATE erreicht (siehe dort). AT(XP,YP) kann - wie im Beispiel gezeigt - auch innerhalb der Ausgabezeile eingesetzt werden, wodurch mit einem PRINT verschiedene Bildschirmpositionen bestimmbar sind.

Ab Version V3.0 wird eine Bereichsunterschreitung (z.B. PRINT AT(0,0) oder LOCATE 0,0) abgefangen, um FATAL-Errors beim Blitter-TOS zu vermeiden.

; (Semikolon)
Dieses Zeichen dient dazu, um verschiedene Ausdrücke miteinander zu verbinden. Der nach diesem Zeichen stehende Ausdruck wird dann direkt an den vorhergehenden angeschlossen. Wenn das Semikolon als Schlußzeichen eingesetzt wird, wird auch der erste Ausdruck des nächsten PRINT-Befehls an den zuletzt ausgegebenen PRINT-Ausdruck angehängt, da Carriage Return und Line Feed (CR/LF = Wagenrücklauf/neue Zeile) in diesem Fall unterdrückt werden.

, (Komma)
Wird das Komma eingesetzt, wird der folgende Ausdruck an den nächsten von 5 Tabulatorpunkten hinter der letzten Ausgabe gesetzt. Diese Punkte haben die X-Positionen 1,17,33,49,65.

' (Apostroph)
Ein Apostroph steht als Leerzeichen. Es werden also in der Bildschirmausgabe überall dort Leerzeichen gesetzt, wo sie durch dieses Zeichen vorgegeben wurden.

Chr$(27);"Z"
Das "Z" steht hier zur Kenntlichmachung dafür, daß ein entsprechender TOS-Befehl eingesetzt werden muß. Das ASCII-Zeichen 27 (ESC) ist eines der TOS-Steuerzeichen, die man bei PRINT einsetzen kann. In diesem Fall wird durch ESC"p" die folgende Schrift invertiert und durch ESC"q" wieder auf Normaldarstellung zurückgeschaltet.

Außerdem wurden String- und numerische Funktionen in die Zeilen eingebaut, um zu zeigen, daß Funktionen problemlos direkt in die PRINT-Ausgabe integriert werden können. Beachten Sie dazu die jeweilige Funktionsbeschreibung unter CHR$(), MIN(), INP(), STRING$(), LEFT$() etc.

Ein direkt angegebener Text ist vor dem ersten Textzeichen mit Anführungszeichen zu versehen (Print "text), da sonst der Text ggfs. als Variable ohne Inhalt interpretiert wird. Folgt auf den Text nichts mehr, setzt der Interpreter die Ausführungszeichen selbsständig.

Ein einzelner PRINT-Befehl ohne nachgestellte Ausdrücke bewirkt den Ausdruck einer Leerzeile (CR/LF).

Wollen Sie innerhalb eines direkt angegebenen Textes Sonderzeichen verwenden, können Sie das auf dieselbe Art erreichen, wie es unter INPUT beschrieben wurde (Control+<a> bzw. <Control>+<s>), wobei allerdings die Steuerzeichen (ASCII 0 - 31) nicht durch PRINT ausgegeben werden können (siehe dazu OUT bzw. OPEN "",#1,"VID:").

Ein kleiner Laufschrift-Trick (Hires/Midres/Lowres):

```
Deffill ,2,4
Pbox 10,10,309,189
@Pstep(3,12,"* Wer nicht sucht, der nicht findet!",0)
Procedure Pstep(Xp%,Yp%,Tx$,Lo%)          ! Als Befehlserweiterung
  ' Xp% = Bildschirm-X-Position (1-79 bzw. 1-39)
  ' Yp% = Bildschirm-Y-Position (1-25)
  ' Tx$ = Laufschrift-Text (max. 79 bzw. max. 39 Zeichen)
  ' Lo% = Durchlauf-Anzahl
  '       wenn 0, dann unendlich (Abbruch durch Taste)
  Local Xt%,Yt%,Xl%,Yl%,Xr%,Yr%,Cnt%,Count%,Backscreen$
  Xt%=2-Sgn(Xbios(4))                      ! X-Auflösungsteiler
  Yt%=Min(2,3-Xbios(4))                    ! Y-Auflösungsteiler
  Xp%=Max(1,Min(79/Xt%,Xp%))               ! X-Position begrenzen
  Yp%=Max(1,Min(25,Yp%))                   ! Y-Position begrenzen
  Tx$=Left$(Tx$,Min(Len(Tx$),(79-(40*(Xt%-1)))-(Xp%-1))) ! Text
  '                                        ! auf max. Zeichen begrenzen
  Deffill ,0,0                             ! Hintergrund-Füllmuster
  Xl%=(Xp%-1)*8-1                          ! Linker String-Rand
  Yl%=(Yp%-1)*(16/Yt%)-1                   ! Oberer String-Rand
  Xr%=Xl%+Len(Tx$)*8+1                     ! Rechter String-Rand
  Yr%=Yl%+(16/Yt%)+1                       ! Unterer String-Rand
  Get Max(0,Xl%),Max(0,Yl%),Min(639/Xt%,Xr%),...
                        ...Min(399/Yt%,Yr%),Backscreen$
  '                                        ! Hintergrund sichern
  Pbox Xl%,Yl%,Xr%,Yr%                     ! Hintergrundbox zeichnen
  Do                                       ! Schleifenstart
    Tx$=Right$(Tx$,Len(Tx$)-1)+Left$(Tx$)! Text um 1 Zeichen versetzen
    Print At(Xp%,Yp%);Tx$;                 ! und ausgeben
    Pause 5                                ! Kleine Pause
    Inc Cnt%                               ! Zeichenzähler +1
    If Cnt%=Len(Tx$) And Lo%>0             ! Text einmal durch?
      Clr Cnt%                             ! Zeichenzähler auf Null
      Inc Count%                           ! String-Zähler +1
    Endif
    Exit If Lo%=0 And (Inkey$>"" Or Mousek)    ! Abbruch, wenn
    '                                          ! Taste und Lo%=0
    Exit If Max(1,Lo%)>0 And Count%=Max(1,Lo%) ! Abbruch, wenn
    '                                          ! Lo%>0 und Zähler=Lo%
```

```
    Loop
    Put Max(0,Xl%),Max(0,Yl%),Backscreen$  ! Hintergrund restaurieren
  Return
```

Eine Variante des Print-Befehls ist PRINT #. In diesem Fall folgt auf das Nummernzeichen eine Zahl, die den anzusprechenden Datenkanal angibt. Wenn Sie also in eine geöffnete Diskettendatei etwas hineinschreiben möchten, ist das hiermit möglich (z.B. Print#1;"BASIC"). Die oben erwähnten Formatierungszeichen können auch hier verwendet werden.

Escape-Sequenzen - auch VT52-Steuersequenzen genannt :

(als Funktionen definiert -> Aufruf: Print @Vtxx$;)

```
Deffn Vt01$=Chr$(27)+"A"
Deffn Vt02$=Chr$(27)+"B"
Deffn Vt03$=Chr$(27)+"C"
Deffn Vt04$=Chr$(27)+"D"
Deffn Vt05$=Chr$(27)+"E"
Deffn Vt06$=Chr$(27)+"H"
Deffn Vt07$=Chr$(27)+"I"
Deffn Vt08$=Chr$(27)+"J"
Deffn Vt09$=Chr$(27)+"K"
Deffn Vt10$=Chr$(27)+"L"
Deffn Vt11$=Chr$(27)+"M"
Deffn Vt12$(X%,Y%)=Chr$(27)+"Y"+Chr$(X%+31)+Chr$(Y%+31)
Deffn Vt13$(F%)=Chr$(27)+"b"+Chr$(F%)
Deffn Vt14$(F%)=Chr$(27)+"c"+Chr$(F%)
Deffn Vt15$=Chr$(27)+"d"
Deffn Vt16$=Chr$(27)+"e"
Deffn Vt17$=Chr$(27)+"f"
Deffn Vt18$=Chr$(27)+"j"
Deffn Vt19$=Chr$(27)+"k"
Deffn Vt20$=Chr$(27)+"l"
Deffn Vt21$=Chr$(27)+"o"
Deffn Vt22$=Chr$(27)+"p"
Deffn Vt23$=Chr$(27)+"q"
Deffn Vt24$=Chr$(27)+"v"
Deffn Vt25$=Chr$(27)+"w"
```

Funktionsaufruf: PRINT @Vtx$;

@Vt01$
Cursor eine Zeile nach oben; wenn in oberster Zeile, stehenbleiben.

@Vt02$
Cursor eine Zeile nach unten; wenn in letzter Zeile, stehenbleiben.

@Vt03$
Cursor ein Zeichen nach rechts; wenn in letzter Spalte, stehenbleiben.

@Vt04$
Cursor ein Zeichen nach links; wenn in erster Spalte, stehenbleiben.

@Vt05$
Clearscreen/Cursor auf Home setzen.

@Vt06$
Cursor auf Home setzen (wie PRINT AT(1,1)).

@Vt07$
Cursor eine Zeile nach oben, wenn in oberster Zeile, dann Screen 1 Zeile abwärts.

@Vt08$
Bildschirm ab Cursor löschen.

@Vt09$
Zeile ab Cursor löschen.

@Vt10$
Zeile an Cursor-Position einfügen und Rest-Screen 1 Zeile abwärts.

@Vt11$
Cursor-Zeile löschen und Rest-Screen nachziehen.

@Vt12$(X,Y)
Cursor auf Position X,Y setzen (wie PRINT AT(X,Y)).

@Vt13$(F)
Cursor-Farbe F setzen (Hires: 0 - 1/Midres: 0 - 3/Lowres: 0 - 15).

@Vt14$(F)
Zeichenhintergrundfarbe F setzen (Hires: 0 - 1/Midres: 0 - 3/Lowres: 0 - 15).

@Vt15$
Bildschirm bis Cursor löschen.

@Vt16$
Cursor anschalten.

@Vt17$
Cursor ausschalten.

@Vt18$
Position des Cursors speichern.

@Vt19$
Cursor auf gespeicherte Position setzen.

@Vt20$
Zeile löschen (Restscreen nicht nachziehen).

@Vt21$
Zeile bis Cursor löschen.

@Vt22$
Reversmodus ein (weiß auf schwarz).

@Vt23$
Reversmodus aus (schwarz auf weiß).

@Vt24$
Cursor nach Erreichen des Zeilenendes auf nächsten Zeilenanfang setzen (Zeilenüberlauf).

@Vt25$
Cursor soll am Zeilenende stehenbleiben.

PRINT USING { P USING } Daten formatiert ausgeben

PRINT USING "format",Expr [,Var,...] [;]
PRINT USING Format$,Expr [,Var,...] [;]
PRINT #Kanal,USING "format",Expr [,Var,...] [;]

Durch diesen Befehl kann ein Ausgabeformat für Werte und Strings bestimmt werden. Ihm folgt als erstes ein String oder eine String-Variable, in der das gewünschte Format angegeben wird. Als nächstes werden nach einem Komma die Ausdrücke oder Variablen übergeben, die diesem Format entsprechen sollen. Werden mehrere Ausdrücke angegeben, müssen diese durch weitere Kommas getrennt werden. Als Formatierungszeichen sind vorgesehen:

\# Platzhalter für eine Ziffer:

```
Print Using "######",Int(31421/3)
Ausgabe: 10473
```

. Position des Dezimalpunktes:

```
Print Using "######.####",31421/4
Ausgabe:  7855.2500
```

\+ Ausgabe auch des positiven Vorzeichens:

```
Print Using "+######.####",31421/4
Ausgabe:+  7855.2500
```

\- Platzhalter für negatives Vorzeichen:

```
Print Using "-#####.######",31421/-4
Ausgabe:- 7855.250000
```

* Füllzeichen für alle angegebenen Vorkommastellen, die nicht von dem auszugebenden Wert belegt werden. Sonst wie #.

```
Print Using "*######.####",31421/1.4
Ausgabe:**22443.5714
```

Hinter dem Dezimalpunkt verwendet, werden so viele * ausgegeben, wie angegeben sind und die auszugebende Zahl wird real auf die gewünschte Stelle gerundet.

```
Print Using "*######.#*****",31421/1.4
Ausgabe:**22443.6****
```

$ Voranstellung eines $:

```
Print Using "$######.##",31421/1.4
Ausgabe: $22443.57
```

, Einfügen eines Kommas (Tausendertrennung):

```
Print Using "##,###,###.###",3142*2781.71
Ausgabe: 8,740,132.820
```

^^^^ Ausgabe im Exponentialformat. Führende # stehen hier für die Stellen des Basis-Anteils und ^ für die Exponentenstellen (E+xxxx). Überflüssige Basis-Stellen werden mit 0 gefüllt. In V3.0 wird der Exponent den Vorkommastellen angepaßt.

```
Print Using "#.########^^^^",13711*64
Ausgabe:8.77504000E+05
```

! Das erste Zeichen eines Strings wird ausgegeben:

```
Print Using "!sing !n !FA","Uhu","igitt","Gaga"
Ausgabe:Using in GFA
```

& Gesamt-String wird ausgegeben:

```
Print Using "&hausen","Enten"
Ausgabe:Entenhausen
```

\..\ Ausgabe von sovielen Zeichen des Strings, wie Länge von \..\ (inkl. Backslashs):

```
Print Using "\..\ingen","Hattu Möhren?"
Ausgabe:Hattingen
```

_ (Tiefstrich) Interpretiert das hierauf folgende Using-Formatzeichen nicht als solches, sondern gibt es als ASCII-Zeichen aus.

```
Print Using "Channel _### _\ &",44,"XYZ"
Ausgabe:Channel #44 \ XYZ
```

Die Formatvorgabe und die String- und/oder Werte- und/oder Ausdrucksliste kann in beliebiger Reihenfolge gegliedert sein, solange die Parametertypen in ihrer Reihenfolge der Formatvorgabe entsprechen. Wird die Reihenfolge nicht korrekt eingehalten, bzw. trifft der Interpreter auf unlogische Formate, wird ein Fragezeichen an der betreffenden Stelle ausgegeben. Bei numerischen Formaten wird dem ausgegebenen Wert ein Prozentzeichen vorangestellt, wenn die Stellenanzahl des Wertes das dafür vorgesehene Format überschreitet. Werden außerdem bei numerischen Werten im Format weniger Nachkommastellen angegeben, als vorhanden sind, werden die übrigen Nachkommastellen integriert.

In V3.0 haben Sie die Möglichkeit, zwischen Dezimalpunkt und -komma, zu wählen (siehe MODE).

Durch Nachstellen eines Semikolons (Print Using "....",Expr,Liste....;) kann - wie bei PRINT - die Ausgabe von CR/LF unterdrückt werden.

Mit Angabe eines Datenkanals (Print #1,Using...) können die Ausgaben auch auf diesen Kanal umgeleitet werden. Die oben angeführten Syntaxregeln sind auch dann gültig.

WRITE { WR } Daten ausgeben

WRITE [#Kanal,] ["Text" [,Var,Expr;...]]

Ein Ausgabe-Befehl, der in erster Linie zur Datenspeicherung in sequentiellen Dateien gedacht ist. Dieser Befehl kann aber auch zur Text-, bzw. Datenausgabe auf dem Bildschirm verwendet werden. Er ist dem PRINT-Befehl sehr ähnlich, hat jedoch einen anderen syntaktischen Aufbau. Außerdem werden hier die Anführungszeichen eines übergebenen Strings (Ausdruck oder Variable) sowie die Kommas, mit denen hier die einzelnen Ausdrücke voneinander getrennt werden, ebenfalls ausgegeben.

Seinen eigentlichen Sinn zeigt dieser Befehl jedoch erst, wenn mit einem INPUT#-Befehl mehrere Werte oder Strings gleichzeitig aus einer

Datei eingelesen werden sollen. Dazu müssen die Einzeldaten durch Kommata voneinander getrennt sein. Da WRITE# Kommata an den entsprechenden Platz in der Datei schreibt, können die Daten beim Einlesen mit INPUT# unterschieden und den dabei angegebenen Variablen zugeordnet werden.

Beispiel:

```
Open "O",#1,"Friends"               ! Datei zur Ausgabe öffnen
Restore N_amen                      ! Data-Zeiger setzen
For I%=1 To 4                       ! 4 Zeilen
  Read Name$,Beruf$,Telefon$        ! Je 3 Datas
  Write #1,Name$,Beruf$,Telefon$ ! in die Datei schreiben
Next I%                             ! Nächste Zeile
Close #1                            ! Datei schließen
N_amen:
Data " Elizabeth "," Königin        "," London/112233       "
Data " Kashogghi "," Milliardär     "," Riad/1.000.000.000  "
Data " Boris     "," The lost Winner "," Leimen/666666      "
Data " Yeti      "," Schneemensch   "," Himalaya/XY-ungelöst "
Open "I",#1,"Friends"               ! Datei zum Einlesen öffnen
Print "Meine besten Freunde:";Chr$(13);Chr$(10)    ! Bla...
Print "Datei-Inhalt ohne Format:";Chr$(13);Chr$(10) ! ...bla
A$=Input$(Lof(#1),#1)               ! Kompletten Datei-Inhalt lesen
Print A$                            ! Unformatiert ausgeben
Seek #1,0                           ! File-Pointer wieder auf Anfang
Print "Mit WRITE ausgegeben:;Chr$(13);Chr$(10)
For I%=1 To 4                       ! 4 Zeilen
  Input #1,N.ame$,B.eruf$,T.elefon$ ! Je 3 Ausdrücke
  Write N.ame$,B.eruf$,T.elefon$ ! mit WRITE ausgeben
Next I%
Seek #1,0                           ! File-Pointer wieder auf Anfang
Print Chr$(10);"Mit PRINT ausgegeben:;Chr$(13);Chr$(10)
For I%=1 To 4                       ! 4 Zeilen
  Input #1,N.ame$,B.eruf$,T.elefon$ ! Je 3 Ausdrücke
  Print N.ame$,B.eruf$,T.elefon$ ! mit PRINT ausgeben
Next I%
Close #1                            ! Datei schließen
Void Inp(2)                         ! Auf Taste warten
```

5.3 Bildschirmoperationen

Version 3.0

HTAB {HT} — Aktuelle Cursor-Spalte bestimmen

HTAB Spalte

Ein Befehl, der vor allem der Kompatibilität zu anderen BASIC-Dialekten und der Zusammenarbeit mit CRSCOL dient (TOS-Bildschirm-Organisation siehe bei PRINT).

Ab Version 2.02

LOCATE { LOCAT } Cursor positionieren

LOCATE S,Z

Positioniert den TOS-Cursor auf Spalte "S" und Zeile "Z". Von der Syntax-Kontrolle wird dieser Befehl in PRINT AT(S,Z); umgewandelt. Weiteres siehe dort.

POS() Cursor-Spalte ermitteln

Var=POS(Dummy)

POS liefert Ihnen die aktuelle Cursor-Spalte. Es muß ein, in Klammern nachgestelltes numerisches Scheinargument angegeben werden. Diese Zahl kann beliebig gewählt werden und hat keinen weiteren Einfluß auf die Funktion.

Da String-Konstrukte mit mehr als 80 (Hires/Midres) bzw. 40 (Lowres) Zeichen ausgegeben werden können und eine TOS-Bildschirmzeile 256 Zeichen aufnehmen kann, kann sich der zurückgegebene Wert im Bereich von 0 - 255 bewegen. Die durch POS ermittelte Spaltenzahl differiert im Bildschirmbereich um -1 von der durch CRSCOL gelieferten Spalte.

Beispiel:

```
Print At(1,1);Pos(0);Spc(20);Pos(0)
        Ausgabe : 0                     21
Print Space$(4000);Pos(0)
        Ausgabe : (4000 Leerzeichen)160
```

Der Wert 160 im Beispiel errechnet sich aus:

```
4000 Mod 256 = 160
```

Ist in der letzten Ausgabe eines der Steuerzeichen CHR$(8) (Backspace) oder CHR$(13) (Carriage Return) enthalten gewesen, haben diese Einfluß auf die POS()-Position:

```
Backspace        (BS=Chr$(8))  Vermindert POS() um 1
Carriage Return  (CR=Chr$(13)) Setzt POS() auf Null
```

SPC() — Leerzeichen ausgeben

PRINT SPC(Anz)
PRINT [Ausdrücke;Werte;etc.;] SPC(Anz) [;etc.]

SPC ist ein Befehl, der nur im Zusammenhang mit PRINT verwendet werden kann. Der in Klammern angegebene Ausdruck Anz steht für die Anzahl an Leerzeichen (SPaCe = 0 - 255), die an der aktuellen Cursor-Position ausgegeben werden sollen.

Beispiel:

```
Print "==>";Spc(20);"Ende"
     Ausgabe : ==>                    Ende
```

Nicht möglich ist:

```
A$="==>"+Spc(20)+"Ende"
```

Es wird ein Syntaxfehler angezeigt. Für solche Konstrukte eignet sich SPACE$(20).

TAB() — Tabulator setzen

TAB(Position)
PRINT [Ausdrücke;Werte;etc.;] TAB(Anz) [;etc.]

Es kann eine Tabulatorposition bestimmt werden, an welcher der Cursor dann positioniert wird. Diese Position kann im Bereich von 0 bis 255 liegen. Größere Werte werden mit MOD 256 auf diesen Bereich zurückgerechnet. Befindet sich der Cursor in einer Zeile hinter der zuletzt angegebenen Tabulatorposition und der Wert einer sich anschließenden TAB-Anweisung liegt zwischen 256 und der aktuellen Cursor-Position, so wird dieser Tabulator in derselben Zeile ausgeführt. Ist der TAB-Wert kleiner als die aktuelle Cursor-Position, wird die Anweisung in der nächsten Zeile ausgeführt. TAB ist ebenso wie SPC nur in Verbindung mit PRINT ausführbar. String-Konstrukte mit TAB sind nicht möglich.

Beispiel:

```
For J%=0 To 11                 ! 12 mal
  Restore T_ext                ! Data-Zeiger setzen
  For I%=1 To 2                ! 4 Datas...
    Read A$,A%                 ! ...lesen...
    Print Tab(J%*6);A$''''''A% ! ... und ausgeben
```

```
  Next I%                       ! Nächstes Data
Next J%                         ! Nächste Position
T_ext:
Data GFA-,1,BASIC,2
```

Version 3.0

VTAB {VT} Aktuelle Cursor-Zeile bestimmen

VTAB Zeile

Ein Befehl, der vor allem der Kompatibilität zu anderen BASIC-Dialekten und der Zusammenarbeit mit CRSLIN dient (TOS-Bildschirm-Organisation siehe bei PRINT).

5.4 Diskettenoperationen

Bei allen Diskettenoperationen, denen ein Dateiname zu übergeben ist, besteht die Möglichkeit, einen Suchpfad zu definieren, über den der Dateizugriff ausgeführt werden soll.

- Laufwerkbezeichnung: Buchstabe mit Doppelpunkt:

A: Für das erste Laufwerk
B: Für das zweite Laufwerk
C: Für evtl. vorhandene RAM-Disk
D: Für evtl. vorhandenes ROM-Modul
H: Für Hard-Disk

- Fragezeichen innerhalb des Dateinamens sind Platzhalter für einzelne Buchstaben (auch "Wildcards" genannt). Es werden alle Dateien angesprochen, die bis auf die Fragezeichenpositionen mit dem angegebenen Dateinamen übereinstimmen.
- Sternchen im Dateinamen sind Platzhalter für einen ganzen Bereich von Zeichen.

Der Suchpfad hat folgende hierarchische Struktur:

```
Station:\ggfs. Ordner\ggfs. Unterordner\Dateiname.Extension
   ^           ^               ^                ^
   !           !               !                !
Direktory -> Sub-Direktory -> SubSub-Direktory ->  File
```

Die Anzahl der Sub-Dir's ist nicht begrenzt, soweit die File-Allocation-Table (FAT) der Diskette oder Hard-Disk genügend Platz bietet. Mehr als drei Ebenen sind jedoch aus Gründen der Übersichtlichkeit nicht ratsam.

Die Stationsangabe kann entfallen. Es wird dann auf der aktuellen Station gesucht. Die Ebenen sind im Pfad durch das Backslash-Zeichen (\) zu trennen. Wird eine Extension (max. 3 Zeichen) angegeben, muß sie durch einen Punkt vom Dateinamen (max. 8 Zeichen) getrennt werden.

Beispiele:

```
B:\UTILITY\OUTPUTS\DRUCKE.WAS
```

Sucht auf Disk B im Unterordner OUTPUTS des Ordners UTILITY nach der Datei DRUCKE.WAS.

```
B:\UTILITY\*.*
```

Sucht auf Disk B im Ordner UTILITY nach der nächsten beliebigen Datei.

```
B:\UTILITY\OUTPUTS\*.WAS
```

Sucht auf Disk B im Unterordner OUTPUTS des Ordners UTILITY nach der ersten Datei mit der Extension WAS.

```
\*.WAS
```

Sucht auf der aktuellen Station im Haupt-Direktory nach der ersten Datei mit der Extension WAS.

```
\UTILITY\DR????
```

Sucht auf der aktuellen Station im Ordner UTILITY nach der ersten Datei mit einem 6-Zeichen-Namen, der mit DR beginnt.

```
A:\DR????.WAS
```

Sucht auf Disk A im Hauptdirektory nach der ersten Datei mit einem 6-Zeichen-Namen, der mit DR beginnt und die Extension WAS trägt.

```
A:\D*.*
```

Sucht auf Disk A im Haupt-Direktory nach der ersten Datei, deren Name mit D beginnt (mit oder ohne Extension).

```
\UTILITY\*KE.*S
```

Sucht auf der aktuellen Station im Ordner UTILITY nach der ersten Datei, deren Name mit KE und der Extension mit S endet.

Der Pfadname kann unter GFA-BASIC in den meisten Fällen als String-Ausdruck, als String-Variable oder als Kombination von beidem übergeben werden.

Da die Pfadanalyse zu einem der häufig auftretenden Probleme zählt, folgt nun ein kleines Utility, daß Ihnen die Arbeit dabei abnimmt.

Beispiel 1:

```
A$="\Ordner\Datei.Dat"        ! Beliebiger Pfad
Print "Vorher  : "
Print ,A$                     ! Anzeigen
@Path(0,A$,"\",*A$,*B$,*F$) ! Nach erstem Backslash suchen
@Path(0,B$,".",*B$,*C$,*F$) ! Punkt im Dateinamen suchen
' ==> Die Prozedur Path finden Sie im
'     Library-Teil des RAMKART-Programms.
Print "Nachher : "
Print ,A$                     ! Pfad
Print ,B$                     ! Dateiname
Print ,C$                     ! Extension
```

Beispiel 2:

```
A$="**Dies*ist-ein¦beliebiger.Text-String!" !String-Vorgabe
Print "Vorher  : "
Print ,A$                     ! Anzeigen
Print "Nachher : "
Print Spc(11);"Teil 1";Spc(33);"Teil 2      Suchzeichen"
Print String$(75,"-")
Repeat                        ! Schleife
  M$=A$                       ! Alte Vorgabe merken
  @Path(0,A$,"*-!.¦",*A$,*B$,*F$) ! Aufruf m. Suchzeichen-Liste
  ' ==> Die Prozedur Path finden Sie im
  '     Library-Teil des RAMKART-Programms.
  Inc I%                      ! Schrittzähler
  Print "Schritt ";I%''A$;Tab(50);B$;Tab(64);F$ ! Ergebnis
Until A$="" Or A$=M$          ! Abbruch, wenn 1.Teil = "" oder
'                             ! wenn 1.Teil = Vorgabe-String
```

Im Folgenden wird Ihnen öfter der Begriff "#Kanal" begegnen. Damit ist bei Datei-relativen Diskettenoperationen der Identifikator (0 - 99) der jeweils angesprochenen Datei gemeint.

BLOAD { BL } Datei in Speicherbereich laden

BLOAD "Dateiname" [,Start]

Mit BLOAD kann eine beliebige Datei komplett vom Festspeicher (Diskette/Hard-Disk/RAM-Disk) an eine beliebige RAM-Adresse (> 2047) geladen werden.

In "Dateiname" wird die Dateibezeichnung (ggfs. inkl. Pfad) übergeben und durch den Parameter "Start" wird die Adresse angegeben, ab welcher die Daten abgelegt werden sollen. Wird "Start" ausgelassen, wird jene Adresse als Ziel verwendet, welche beim letzten BSAVE-Aufruf als Quelle gedient hat. Beide Parameter können auch in Variablen übergeben werden.

Beispiel (als Befehlserweiterung konzipiert):

```
Ext$="PI"+Str$(Xbios(4)+1)     ! Extension '.PIx'
' XBIOS(4) liefert die aktuelle Auflösungskennziffer:
'          0 = Lowres    ->   Extension = PI1
'          1 = Midres    ->   Extension = PI2
'          2 = Hires     ->   Extension = PI3
Fileselect "\*."+Ext$,"Degaspic."+Ext$,A$
@Degload(A$,*Pic$,*Pal$,*Res%) ! Aufruf
If Res%<>Xbios(4)              ! Falsche Auflösung
  Alert 3,"Bild hat falsche Auflösung !",1,"Return",B%
Else
  Void Xbios(6,L:Varptr(Pal$)) ! Farb-Palette installieren
  ' XBIOS(6,L:Adr%) überträgt ab der angegebenen Adresse
  '                'Adr%' 32 Bytes in die Farbregister
  Sput Pic$                    ! Bild ausgeben
Endif
Procedure Degload(P.nm$,P.ad%,C.ad%,P.rs%)! Als Erweiterung
  ' P.nm$ = Gewählter Bildname
  ' P.ad% = Pointer auf String-Rückgabevariable, die
  '         nach Abschluß das Bild (32000 Byte) enthält
  ' C.ad% = Pointer auf String-Rückgabevariable, die
  '         nach Abschluß die Farbpalette enthält
  ' P.rs% = Pointer auf Integer-Rückgabevariable, die
  '         nach Abschluß die Auflösungskennung des
  '         geladenen Bildes enthält
  Local Buff$,B.adr%           ! Lokale Variablen
  Buff$=Space$(32034)          ! Bild-Puffer anlegen
  B.adr%=Varptr(Buff$)         ! Puffer-Startadresse
  Bload P.nm$,B.adr%           ! Degas-Bild in Puffer laden
  *P.ad%=Right$(Buff$,32000) ! Bild abschneiden und Rückgabe
  *C.ad%=Mid$(Buff$,3,32)      ! Palette    "      "      "
  *P.rs%=Dpeek(Varptr(Buff$))! Auflösung isolieren und "
Return
```

Aufbau eines Degas-Bildes:

```
Byte 1-2  (Word 1)    : Auflösungkennziffer (Xbios(4))
Byte 3-34 (Word 2-17) : Farbwerte für TOS-Register 0 - 15
Byte 35-32034         : Bilddaten
```

Vor einem BLOAD-Aufruf sollten Sie sicherstellen, daß die zu ladende Datenmenge auch tatsächlich ab der angegebenen Speicherstelle untergebracht werden kann, ohne in wichtige Bereiche hineinzuschreiben. In den meisten Fällen wird dies durch das Einrichten eines ausreichend großen Puffers erledigt.

Wäre die Variable Buff$ im obigen Beispiel kleiner als 32034 Bytes, könnten Sie sich schon mal auf den Weg zum Reset-Knopf machen, da nämlich dadurch der Backtrailer (siehe Variablenorganisation) des Strings zerstört werden würde und das BASIC dann ziellos durch die Wüste läuft. Bei diesem Beispiel wird ein beliebiges Degas-Bild in einen Puffer geladen. Ist die Bild-Auflösung korrekt, wird nach Rückkehr aus der Lade-Routine durch die XBIOS-Funktion 6 Setpalette die Farbpalette installiert und das Bild durch SGET ausgegeben. Anschließend können Sie das Bild jederzeit (solange Pic$ unverändert bleibt) durch SPUT Pic$ wieder auf den Bildschirm holen.

BSAVE { BS } Speicherbereich auf Disk speichern

BSAVE "Dateiname",Start,Anz

Ein sehr komfortabler Befehl, auf den man sicher immer wieder zurückkommen wird. Er bietet die Möglichkeit, eine beliebige Anzahl von Bytes (aus RAM oder ROM) als gesamten Block (BSAVE = BlockSAVE) auf Diskette zu speichern. Es muß der Name der Datei angegeben werden, die diesen Block aufnehmen soll. Nach einem Komma folgt die Adresse (Start > 2047), in welcher das erste Byte des Blocks steht und abschließend wieder nach einem Komma die Anzahl (Anz) der Bytes, die ab dieser Adresse gelesen und gespeichert werden soll.

Beispiel 1:

Der gesamte sichtbare Bildschirmspeicher (XBIOS(2) = Anfangsadresse) wird im sogenannten Doodle- bzw. Standard-Format (32000 Bytes) auf Disk A verfrachtet.

```
Bsave "A:\Screen.Doo",Xbios(2),32000
```

Beispiel 2:

(Als Befehlserweiterung konzipiert) Hier wird der Bildschirmspeicher im Degas-Format auf der aktuellen Station abgelegt:

```
Pic$="Degas.Pi"+Str$(Xbios(4)+1)        ! Dateinamen bilden
@Degsave(Pic$,Xbios(2))                 ! Aufruf
Procedure Degsave(P.name$,P.adrs%)      ! Als Erweiterung
  ' P.name$ = Dateiname (ggfs. inkl. Pfad)
  ' P.adrs% = Adresse, ab welcher die zu sichernden
  '           Bilddaten liegen.
  Local Buff$,I%                        ! Lokale Variablen
  Buff$=Space$(32034)                   ! Puffer anlegen
  Bmove Xbios(2),Varptr(Buff$)+34,32000 ! Bildschirmspeicher
  '                                     ! in Puffer schreiben
  For I%=0 To 15                        ! 16 Farbregister
    Dpoke Varptr(Buff$)+2+I%*2,(Xbios(7,I%,-1) And &H777)
    ' Farbwert lesen und der Reihe nach in Puffer schreiben
    '
    ' XBIOS(7,Reg%,-1) And &H777 liefert den Wert des
    ' angegebenen Farbregisters Reg% (0-15). Dieser
    ' Wert liegt im Bereich 0 (&H0) bis 1911 (&H777).
    '
  Next I%
  Dpoke Varptr(Buff$),Xbios(4)          ! Auflösungskennziffer
  '                                     ! in Puffer schreiben
  Bsave Pic$,Varptr(Buff$),32034        ! Puffer abspeichern
Return
```

CHAIN { CH/V3.0: CHAI } Programm laden (Autostart)

CHAIN "Programmname"

Dieser Befehl ist eigentlich identisch mit LOAD. Allerdings wird das hiermit geladene Programm nach dem Laden automatisch gestartet. Da bei diesem Programmstart - wie sonst auch - alle Variablen und Felder gelöscht werden, können zwischen den "gechainten" Programmen diese nicht ausgetauscht werden. Hier hat man nun folgende Möglichkeiten:

1. Sie können alle wichtigen Daten, die von dem aufrufenden Programm übergeben werden sollen, in eine Datei schreiben (siehe PRINT#/WRITE#), diese am Programmanfang des CHAIN-Programms wieder einlesen (siehe INPUT#) und die Puffer-Datei dann wieder löschen. Der Vorteil ist hier, daß die übergebene Datenmenge nur durch den freien Disk-Speicherplatz begrenzt ist. Beispiel:

```
Programm 1:

A%=1025                  ! Beliebige Variable
B=399.22                 !    "       "
C$="BASIC"               !    "       "
Open "O",#1,"Vars.Dat"   ! Puffer-Datei öffnen
Write #1,A%,B,C$         ! Daten übergeben
Close #1                 ! Datei schließen
Chain "Program2.Bas"     ! Programm 2 aufrufen

Programm 2 (Program.Bas):

Open "I",#1,"Vars.Dat"   ! Puffer-Datei öffnen
Input #1,A%,B,C$         ! Daten einlesen
Close #1                 ! Datei schließen
Kill "Vars.Dat"          ! Puffer-Datei löschen
Print "Variablen aus Programm 1 : ";A%''B''C$
```

2. Sie schreiben die Daten, die Sie übergeben möchten in den 128 Byte großen Kommandoteil der Basepage (Adresse: Basepage + 128). Hierbei ist allerdings die Datenmenge auf 128 Bytes beschränkt und die Daten müssen schnellstmöglich durch das aufgerufene Programm ausgelesen werden, da bei Diskettenzugriffen diese Kommadozeile auch vom Interpreter benutzt wird. Beispiel:

```
Programm 1:

Poke Basepage+128,3           ! Anzahl d. MKL$-Werte
A$="Text-String"              ! Beliebiger Text
Poke Basepage+129,Len(A$)     ! Anzahl d. Textzeichen
A$=Mkl$(A%)+Mkl$(B%)+Mkl$(C%)+A$ ! String bilden
Bmove Varptr(A$),Basepage+130,Len(A$) ! In die
'                             ! Basepage schreiben
Chain "Program2.Bas"          ! Programm 2 aufrufen

Programm 2 (Program2.Bas):

Anz%=Peek(Basepage+128)       ! Werte-Anzahl
Dim A%(Anz%)                  ! Integer-Feld
For I%=0 To Anz%-1            ! MKL$-Daten
  A%(I%)=Lpeek(Basepage+130+I%*4)! auslesen
Next I%
A$=Space$(Peek(Basepage+129))! String-Puffer
Bmove Basepage+130+I%*4,Varptr(A$),Peek(Basepage+129)
'                                 ! Textanteil in eine
'                                 ! String-Variable holen
Print "Variablen aus Programm 1 :"
For I%=1 To Anz%
  Print A%(I%-1)
Next I%
Print A$
```

In diesem Beispiel legt das aufrufende Programm nach einem vorgewählten Schema (das Ihnen überlassen bleibt) zuerst drei MKL$-Werte und daran anschließend eine Text-String in der Basepage ab. Da das aufrufende Programm dann natürlich "weiß", wie die Basepage (bzw. die zweite Hälfte davon) zu entschlüsseln ist, können hier die Daten wieder extrahiert und weiterverwendet werden.

3. Sie verfahren nach dem gleichen Prinzip, wie unter Punkt 2 beschrieben, nur daß Sie die Daten nicht in der Basepage, sondern hinter dem Bildschirmspeicher puffern. Direkt hinter dem Bildschirmspeicher (Startadresse: XBIOS(2)+32000) befindet sich nämlich ein Speicherbereich von 767 Bytes, der vom Betriebssystem in keiner Weise genutzt wird. Er ist einfach leer. Es sei denn, irgendwelche Accessories (sehr unwahrscheinlich) oder Autostart-Programme (schon wahrscheinlicher) benutzen diesen Bereich zur Datenablage.

   Ist das nicht der Fall, können Sie diesen Bereich nach eigenem Gutdünken verwenden. Prüfen Sie vorher, ob sich irgendwelche Daten dort befinden:

```
A$=Space$(767)
Bmove Xbios(2)+32000,Varptr(A$),767
If String$(767,0)=A$
.... hier kommen Sie nur hin, wenn der
     Bereich absolut leer ist....
```

Im Interpreterbetrieb wird durch CHAIN auch der BASIC-Arbeitsspeicher samt Inhalt (aufrufendes Programm) vorher gelöscht. Wenn Sie kein PSAVE-geschütztes Programm aufgerufen haben, finden Sie nach Programmende das nachgeladene Programm auch im Editor.

Bei Compilaten wird durch CHAIN ein direkt ausführbares .PRG-, .TOS- oder .TTP-Programm gestartet. Vor dessen Ausführung wird der Name des aufgerufenen Programms an die AES-Funktion shell_write übergeben und das aufrufende Programm beendet. Es ist danach nicht mehr aufrufbar und der Speicher wird freigegeben. Die zweite Hälfte der Basepage (Bytes 129 - 256) wird durch die AES-Funktion shell_write (in GFA-BASIC) als Kommandozeile in die Basepage des aufgerufenen Programms geschrieben. Es ist also auch hier möglich, diese 128 Bytes dazu zu verwenden, Informationen, Variableninhalte etc. nach dem oben beschriebenen Schema an das aufgerufene Programm zu übergeben. Der Unterschied zu EXEC besteht

darin, daß nach Abschluß des aufgerufenen Programms das System nicht zum Desktop, sondern zum residenten Aufrufer zurückkehrt.

CHDIR { CHD } Ordner wechseln

CHDIR "Ordner"

Hiermit kann auf der aktuellen Diskette ein Ordner geöffnet werden, bzw. falls schon einer geöffnet ist, wird dieser vorher geschlossen und der angegebene Ordner geöffnet. Dieser Ordner gilt im weiteren Programm (bis zur nächsten Änderung) als Default-Ordner, d.h. er wird bei Diskettenzugriffen als aktuelles Verzeichnis voreingestellt. Dem Befehl wird einfach der Name des Ordners übergeben, der geöffnet werden soll (Pfadstruktur siehe Kapitelanfang). Diese Bezeichnung kann wieder entweder in einer Variablen enthalten sein (Chdir Ordner$), oder als Textkonstante übergeben werden (Chdir "UTILITY").

Besteht der Ordnername nur aus einem Backslash (\), greift CHDIR danach nicht auf einen Ordner, sondern auf das Haupt-Directory zu.

In V3.0 kann immer dann, wenn ein übergeordneter Ordner vorhanden ist, die Übergabe dessen Namens gespart werden, indem man dafür '..' einsetzt. Angenommen, der aktuelle Ordner 'Ordner1' liegt im Ordner 'Chef_Ord.ner'. Ebenfalls in diesem Über-Ordner befindet sich noch ein weiterer Ordner 'Ordner2', zu welchem gewechselt werden soll.

In V2.xx müßte nun CHDIR "chef_ord.ner\ordner2" geschrieben werden. Diese Angabe verkürzt sich durch '..' zu CHDIR "..\Ordner2". Zusätzlich kann für den aktuellen Ordnernamen '.' verwendet werden.

CHDRIVE { CHDR } Aktuelles Laufwerk bestimmen

CHDRIVE Laufwerk
CHDRIVE Pfad$ **(nur für V3.0)**

Es kann ein Laufwerk zum aktuellen Laufwerk bestimmt werden.

1 = Laufw. A/2 = Laufwerk B/... 15 = Laufw. O.

Wird bei künftigen Pfadangaben kein Laufwerk bestimmt (z.B. List "\Ordner\Programm.Lst"), wird von nun ab das durch CHDRIVE aktualisierte Laufwerk angesprochen.

In Version V3.0 kann in Pfad$ ein Textausdruck angegeben werden, dessen erstes Zeichen als Stationsbestimmung interpretiert wird (z.B. CHDRIVE "A:*.*").

DFREE() Freien Disketten-Speicherplatz ausgeben

Var=DFREE(Station)

Bei Verwendung dieses Befehls wird der momentan freie Speicherplatz der angegebenen Station ermittelt und zurückgegeben. Wie schon erwähnt, werden den Laufwerken Nummern zugewiesen, über die sie bei Befehlen wie diesem zu erreichen sind:

0 = Aktuelles Laufw./1 = Laufw. A/... 15 = Laufw. O

Wie alle anderen Funktionen kann auch diese auf beliebige Art eingesetzt werden.

Beispiele:

```
A%=Dfree(0)
Print "Freier Speicher auf Disk A: ";Dfree(1)
If Dfree(2)<32000
  Alert 2,"Speicher auf Disk B ??",1,"Abbruch",B%
Else
  Bsave "B:\Picname.Pic",Xbios(2),32000
Endif
```

DIR Directory ausgeben

DIR ["Pfad"] [TO "Datei"]

Durch DIR lassen sich auf verschiedene Weise Inhaltsverzeichnisse entweder einer beliebigen Diskettenstation oder eines bestimmten Ordners ausgeben. Dabei ist es auch möglich, diese Directory in eine bestimmte Diskettendatei zu schreiben oder auf dem Drucker ausdrucken zu lassen. Zur Pfadangabe können sämtliche am Kapitelanfang beschriebenen Spezifikationen verwendet werden.

Beispiele:

```
Dir
```

Gibt **alle** Dateien der aktuellen Station, die sich außerhalb befinden, auf dem Bildschirm aus.

```
A$="A:\UTILITY\"
B$="D????"
Dir A$+B$+".EXT"
```

Listet alle Dateien der Disk A, die sich im Ordner UTILITY befinden, deren Name mit dem Buchstaben D beginnt, gleichzeitig fünf Buchstaben hat und die Extension EXT trägt.

```
Dir "\*.EXT" To "B:\DIRECTOR\SUBDIR_1.SUB"
```

Schreibt **alle** Dateien mit der Extension EXT, die sich auf der aktuellen Diskette im Haupt-Direktory befinden, in eine neue Datei mit dem Namen Subdir_1.Sub, die in dem schon bestehenden Ordner Director abgelegt wird.

Würde im letzten Beispiel statt der Pfadbezeichnung hinter TO der Ausdruck "LST:" als Zieldatei angegeben, würde die Ausgabe der Dateinamen auf dem Drucker erfolgen.

DIR$() — Aktuellen Ordnernamen ermitteln

Var$=DIR$(Laufwerk)

Mit dieser Funktion kann der Name des zur Zeit geöffneten Ordners einer bestimmten Station ermittelt werden. "Laufwerk" steht für die Station, auf der nachgesehen werden soll:

0 = Aktuelles Laufw./1 = Laufw. A/... 15 = Laufw. O

Ist kein Ordner geöffnet, wird ein Leer-String zurückgegeben.

Beispiele:

```
Print Dir$(0)   ! Namen des aktuellen Ordners der aktuellen
'               ! Station auf dem Bildschirm ausgeben
A$=Dir$(2)      ! Namen des aktuellen Ordners der Disk B
'               ! an die Variable A$ übergeben
```

EXIST() — Existenz einer Datei prüfen

Var=EXIST(Dateiname)

Ermittelt, ob die Datei "Dateiname" vorhanden ist. Es wird entweder eine 0 (= FALSE -> nicht vorhanden) oder -1 (= TRUE -> vorhanden) zurückgegeben. Die Angabe eines Suchpfades erfolgt ggfs. gemäß der am Kapitelanfang beschriebenen Struktur.

Beispiel:

```
If Exist("B:\Datei.Dat")        ! Datei.Dat auf B?
  Open "I",#1,"B:\Datei.Dat"    ! Ja, dann öffnen
  '...weiteres Programm
Else
  Alert 1,"B:\Datei.Dat|existiert nicht!",1,"Abbruch",B%
Endif
```

Version 3.0

FGETDTA() Disk-Transfer-Adresse ermitteln

Var=FGETDTA()

Liefert die Startadresse des 44 Byte großen Disk-Transfer-Puffers (die Disk-Transfer-Adresse). Verschiedene Diskettenzugriffe (DIR, FILES, EXIST, FSFIRST etc.) verwenden diesen Puffer zur Ablage des aktuell gelesenen Dateinamens und seiner Attribute. Bei BASIC-Start liegt die DTA voreingestellt auf BASEPAGE+128. Die DTA läßt sich in GFA-BASIC durch LPEEK(BASEPAGE+32) (V2.xx) bzw. {BASEPAGE+32} (V3.0) ermitteln.

```
  Offset    Bedeutung

  0 - 20    21 Bytes für GEMDOS reserviert.
  21        Datei-Attribut (1 Byte)
            PRINT "Attribute : ";BYTE{FGETDTA()+21}
  22        Uhrzeit (2 Byte -> GEMDOS-Format).
            sek%=(CARD{FGETDTA()+22} AND 31)*2
            min%=(CARD{FGETDTA()+22} DIV 32) AND 63
            std%=(CARD{FGETDTA+22} DIV 2048) AND 31
            PRINT "Zeit : ";std%;":";min%;":";sek%
  24        Datum  (2 Byte -> GEMDOS-Format).
            tag%=CARD{FGETDTA()+24} AND 31
            mon%=(CARD{FGETDTA+24} DIV 32) AND 31
            jhr%=((CARD{FGETDTA()+24} DIV 512) AND 31)+1980
            PRINT "Datum : ";tag%;".";mon%;".";jhr%
            (siehe dazu auch TOUCH#)
  26        Dateilänge (4 Byte).
            PRINT "Länge : ";{FGETDTA()+26}
  30 - 43   Dateiname (max. 14 Byte -> C-String).
            PRINT "Name : ";CHAR{FGETDTA()+30}
Datei-Attribute (FGETDTA()+21):
   0  = Normale Datei (lesen/schreiben)
   1  = Schreibgeschützte Datei
   2  = Versteckte Datei ---.  Diese beiden Dateitypen sind auf
                            |-- dem Desktop nicht sichtbar und
   4  = System-Datei     ---'  werden von FILES, FILESELECT etc.
                               nicht angezeigt. Sie sind jedoch
                               wie jede andere Datei ansprechbar.
   8  = Diskettenname (Volume Label - nicht ansprechbar)
   16 = Ordner (Sub-Directory)
   32 = Archiv-Datei (unbenutzt, außer bei Hard-Disk)
```

```
Von manchen Kopier-Programmen wird dieses 7.Bit des
Attributs gesetzt, sobald von der betreffenden
Datei ein Backup angefertigt wurde.
```

Die Attribute einer Datei zu erfahren, ist schon recht nützlich. Interessant wird es jedoch erst, wenn man diese auch frei bestimmen kann. Dies ist über die GEMDOS-Funktion Change Mode (auch Chmod genannt) möglich (weiteres im Kap. 21.2 "GEMDOS"). Ein Beispiel zu FGETDTA() finden Sie unter FSNEXT().

Als V2.xx-Funktion:

```
Print @Fgetdta      ! Mit Aufruf-Zeichen @ und ohne Klammer
Deffn Fgetdta=Gemdos(47)
```

Diese DEFFN-Funktion tut exakt dasselbe wie die V3.0-Funktion.

FILES { V2.0: FILE } — Directory (erweitert) ausgeben

FILES ["Pfad"] [TO "Datei"]

Genügen Ihnen die reinen Dateinamen nicht, die mit DIR ausgegeben werden, können Sie mittels dieses Befehls auch erweiterte Attribute der einzelnen Dateien erfahren. So haben Sie die Möglichkeit, zusätzlich die Größe der Dateien, das Datum und die Uhrzeit ihrer Erstellung in Erfahrung zu bringen. Im übrigen wird dieser Befehl exakt genauso angewendet wie DIR (z.B. FILES listet alle Dateien der aktuellen Disk außerhalb von Ordnern sowie alle Ordner auf). Um die einzelnen Attribute einer bestimmten Datei zu ermitteln, könnte man so vorgehen:

```
Fileselect "\*.*","",F.name$        ! Datei wählen
If Exist(F.name$)                   ! Datei existiert?
  Files F.name$ To "BUFFDAT"        ! FILES -> Puffer-Datei
  Buffer$=Space$(44)                ! String-Puffer bauen
  Bload "BUFFDAT",Varptr(Buffer$)   ! Puffer-Datei in
  '                                 ! String-Puffer laden
  Kill "BUFFDAT"                    ! Puffer-Datei löschen
  F.sz%=Val(Mid$(Buffer$,15,8))     ! Größe extrahieren
  F.tm$=Mid$(Buffer$,23,10)         ! Uhrzeit extrahieren
  F.dt$=Mid$(Buffer$,33,10)         ! Datum extrahieren
  Print "Größe: ";F.sz%''"Zeit: ";F.tm$''"Datum: ";F.dt$
Endif
```

Hier wird ein kleiner Umweg über die Diskette gegangen, da der durch FILES gelieferte String nicht direkt in eine Variable übergeben werden kann. So wird die Infozeile der gewählten Datei erstmal in einer Puffer-Datei zwischengespeichert, um dann mit BLOAD in einen

vorbereiteten String-Puffer geladen zu werden. Nun muß man nur noch die Stellen der Zeile extrahieren, in denen die gewünschten Informationen stehen. Das Datum ist hinterher im DATE$-Format in der Variablen F.dt$, die Uhrzeit im TIME$-Format in der Variablen F.tm$ und die Größe in F.sz% jederzeit verfügbar.

Um allein an die Dateigröße heranzukommen, gibt es eine wesentlich einfachere Methode:

```
Fileselect "\*.*","",F.name$  ! Datei wählen
If Exist(F.name$)             ! Datei existiert?
  Open "U",#99,F.name$        ! Datei öffnen
  F.sz%=Lof(#99)              ! Größe durch LOF ermitteln
  Close #99                   ! Datei schließen
  Print "Dateigröße : ";F.sz% ! Größe ausgeben
Endif
```

In V3.0 geht das - natürlich - alles noch etwas einfacher (siehe FGETDTA() und FSNEXT()).

Version 3.0

FSETDTA() Disk-Transfer-Adresse bestimmen

Var=FSETDTA(Adresse)

Ermöglicht die Bestimmung der Startadresse des Disk-Transfer-Puffers (siehe FGETDTA()). Adresse ist eine beliebige gerade Adresse im User-Bereich (größer 2047 im RAM). Diese Adresse wird von den meisten in GFA-BASIC implementierten GEMDOS-Funktionen (EXIST/DIR/FILES etc.) von nun an als DTA akzeptiert.

Die DTA ist trotzdem nicht unbedingt konstant, da ihre Lage durch einige Befehle, die auf den Festspeicher zugreifen (insbesondere AES-Implementationen, z.B. FILESELECT), verändert worden sein kann.

Wird in Adresse eine ungültige Speicheradresse angegeben oder tritt ein anderer Fehler auf, so enthält die Rückgabevariable Var nach Abschluß eine GEMDOS-Fehlernummer, deren Bedeutung Sie entweder über ERROR Var oder anhand der GEMDOS-Fehlerliste (siehe Kapitel 21.2 "GEMDOS") erfahren können. Ein Beispiel hierzu finden Sie unter FSNEXT().

Als V2.xx-Funktion:

```
Back%=@Fsetdta(Adresse)
Deffn Fgetdta(Adr%)=Gemdos(26,L:Adr%)
```

Diese DEFFN-Funktion tut exakt dasselbe wie die V3.0-Funktion.

Version 3.0

FSFIRST() **Datei suchen**

Var=FSFIRST(Pfad$,Attribute)

Sucht die erste Datei im Direktory, auf welche die in Pfad$ vorgegebenen Suchvorgaben zutreffen, z.B. B:\INFOS\BASIC*.TXT (siehe Kapitelanfang). Zusätzlich können Datei-Attribute (siehe FGETDTA()) angegeben werden. Läßt sich eine Datei nicht in die Attribut-Vorgabe einordnen, wird sie bei der Suche ignoriert. Sollen alle möglichen Datei-Attribute berücksichtigt werden, kann der Wert 63 (Summe aller Attribut-Einzelwerte) angegeben werden.

Wird eine Datei gefunden, werden die unter FGETDTA() beschriebenen Informationen in den Disk-Transfer-Puffer geschrieben und können dort ausgelesen werden. Andernfalls enthält Var nach Abschluß eine GEMDOS-Fehlernummer (sonst 0). Ein Beispiel hierzu finden Sie unter FSNEXT().

Als V2.xx-Funktion:

```
Back%=@Fsfirst(Pfad$+Chr$(0),Attribute)
Deffn Fsfirst(Nm$,Attr%)=Gemdos(78,L:Varptr(Nm$),Attr%)
```

Diese DEFFN-Funktion tut exakt dasselbe wie die V3.0-Funktion. Hier **muß** (!!) allerdings die übergebene Pfadbezeichnung mit einem Null-Byte abgeschlossen werden.

Version 3.0

FSNEXT() **Weitere Datei suchen**

Var=FSNEXT()

Sucht die nächste Datei, auf welche die beim letzten FSFIRST()-Aufruf vorgegebenen Suchkriterien zutreffen. Wird eine weitere Datei gefunden, können auch hier die entsprechenden Daten aus dem Disk-Transfer-Puffer ausgelesen werden (siehe FGETDTA()). Andernfalls enthält Var nach Abschluß eine GEMDOS-Fehlernummer (-33 = Datei nicht gefunden).

Als V2.xx-Funktion:

```
Back%=@Fsnext   ! Mit Aufruf-Zeichen @ und ohne Klammer
Deffn Fsnext=Gemdos(79)
```

Diese DEFFN-Funktion tut exakt dasselbe wie die V3.0-Funktion.
Beispiel:

Wen hat es nicht schon gestört, daß es nur unter großem Kraftaufwand möglich ist, sich ein komplettes Inhaltsverzeichnis seiner Disketten oder Hard-Disk zu erstellen.

Dieses Beispiel liefert Ihnen eine tabellarische Übersicht über alle (ja, wirklich alle!) Dateien einer Station, egal wieviele Ordner und Unter-Unter-Unter...-Ordner vorhanden sind. Möglich wird dies durch eine rekursive Prozedur, die sich ab- und aufwärts durch den Datei-Dschungel bewegt.

Sie hat einen kleinen Mangel, dessen Grund ich nicht feststellen konnte. Wird die Prozedur mehrmals hintereinander für immer dieselbe Station aufgerufen, kann es sein, daß Ordner nicht mehr durchsucht werden. Wird daraufhin die Diskette gewechselt und die Routine für die neue Diskette aufgerufen, ist wieder alles okay. Danach kann dann die Suche auf der vorherigen Station (nochmal Disk wechseln) wiederholt werden.

Beim Studieren des Listings wird Ihnen sicher der Dateizähler Cnt% auffallen. Dieser hat den Zweck, sich die Anzahl der bisher gelesenen Dateien der aktuellen Ebene zu merken. Der Grund dafür liegt darin, daß bei Einsatz von FSFIRST() jedesmal die Suche wieder ganz am Anfang mit der allerersten Datei beginnt.

Da aber die bisher ausgegebenen Dateien bei Rückkehr aus einem Ordner nicht noch einmal beachtet werden sollen, wird immer erst dann wieder ausgegeben, wenn die Anzahl der neu gelesenen Dateien den Index des letzten Ordners erreicht hat. Die Variable Mem$ hat dann die Aufgabe, sich den letzten Ordnernamen zu merken und denselben Ordner beim nächsten Mal zu überspringen.

Eine weitere Besonderheit ist die Variable Dx_level.999%. Sie ist die einzige globale Variable und stellt fest, wann sich das Programm auf der obersten Rekursionsebene befindet. Würde sie nicht verwendet, würde bei jedem rekursiven Prozedur-Aufruf die angegebene Ausgabe-Datei geöffnet und geschlossen, was dann natürlich dazu führt, daß immer nur das Verzeichnis des letzten Ordners dort abgelegt würde.

Diese Raffinesse wurde eingesetzt, um die besonderen Probleme einer Rekursion darzustellen. In diesem speziellen Fall wäre es auch möglich - und auch eleganter -, dem Rekursionsaufruf als Ausgabe-Dateinamen einen Leer-String zu übergeben und die Entscheidung zum Öffnen und Schließen anhand eines gültigen Dateinamens zu treffen. Da aber ein Leer-String hier als Bildschirm-Ausgabe-Flag gewertet wird, müßte dann die Bedingung Else if Fname$="CON:" or fname$="" in die Form Else if Fname$="CON:" umgeändert werden.

Wenn Ihnen die Struktur der Routine nicht auf Anhieb klar wird, dann trösten Sie sich bitte damit, daß eine Rekursion auch für gestandene Profis oft ein Buch mit sieben Siegeln ist.

Die Prozedur erwartet drei Parameter:

Pfad$
Suchpfad für gewünschte Dateien. Wird hier mit einer Extension gearbeitet und ein Ordner hat nicht dieselbe Extension, wird er nicht gefunden und demnach auch nicht durchsucht!

Attr%
Attribut-Wert, den die zu findenden Dateien haben dürfen (siehe FGETDTA()).

Fname$
Name einer Zieldatei. Wird hier ein Leer-String ("") oder "CON:" angegeben, so wird das Verzeichnis auf dem Bildschirm ausgegeben.

"LST:" oder "PRN:" geben auf dem Drucker aus und "AUX:" über den Auxilliary-Port.

```
~Fsetdta(Basepage+128)              ! DTA setzen
Getdir("\*.*",63,"Inhalt.Lst")      ! Alle (!) Dateien suchen
Procedure Getdir(Pfad$,Attr%,Fname$)
  ' Pfad$=Suchpfad/Attr%=Dateiattribut/Fname$=Zieldatei
  If Dx_level.999%=0                ! Oberste Rekursionsebene?
    Fname$=Upper$(Fname$)           ! Zieldateiname nur groß
    If Fname$="PRN:" Or Fname$="LST:" ! Drucker-Ausgabe?
      Open "",#99,Fname$            ! Dann Drucker-Port auf
    Else if Fname$="AUX:"           ! Auxilliary-Ausgabe?
      Open "",#99,"AUX:"            ! Dann AUX:-Port öffnen
    Else if Fname$="CON:" or fname$="" ! Bildschirm-Ausgabe?
      Open "",#99,"CON:""           ! Dann Screen-Port öffnen
    Else                            ! Disk-Datei-Ausgabe?
```

```
    Open "O",#99,Fname$         ! Dann Datei öffnen
  Endif
Endif
Local Tag¦,Mon¦,Jhr&,X%,Cnt%,Td$,D.len%  !--.
Local Pos%,File$,Mem$,Sek¦,Min¦,Std¦        !- Lokale Vars
Local D.att¦,D.uhr&,D.dat&,Depth$,Depth% !--'
Absolute D.att¦,Fgetdta()+21    ! 'D.att¦' --.  Auf die
Absolute D.uhr&,Fgetdta()+22    ! 'D.uhr&'   |- entsprechenden
Absolute D.dat&,Fgetdta()+24    ! 'D.dat&'   |  DTA-Einträge
Absolute D.len%,Fgetdta()+26    ! 'D.len%  --'  setzen
X%=Fsfirst(Pfad$,Attr%)         ! Erste Datei der Ebene suchen
Astring(0,Pfad$,"\",Depth$)     ! Wie viele Backslashe?
' Die Befehlserweiterung Astring finden Sie unter INSTR!
Depth%=Len(Depth$)/2            ! MKI$'s durch 2 = Ebenenindex
Sta:                            ! Label für Rücksprung aus
'                               ! zuletzt bearbeiteter Ebene
Clr Cnt%                        ! Dateizähler klar
While X%=0                      ! Datei gefunden?
  Inc Cnt%                      ! Dateizähler +1
  If Cnt%>Pos%                  ! Zähler größer als
    '                           ! letzter Ordner-Index?
    File$=Char{Fgetdta()+30}    ! Dateinamen lesen
    If File$<>"." And File$<>".." ! <> Subdir-Platzhalter?
      If (D.att¦ And 16) And File$<>Mem$ ! Neuer Ordner?
        Pos%=Cnt%               ! Ordner-Index merken
        Mem$=Pfad$    !file$    ! Bisherigen Pfad merken
        Path1$=Left$(Pfad$,Rinstr(Pfad$,"\")-1) ! Pfadnamen
        '                                       ! isolieren
        Path2$=Right$(Pfad$,Len(Pfad$)-Rinstr(Pfad$,"\")+1)
        '                       ! Suchriterium isolieren
        Print #99,"'";Spc((Depth%)*3);" ORDNER: "; ! Ordner-
        Print #99,Path1$+"\"+File$+Path2$ ! Pfad ausgeben
        Print #99,"'";Spc((Depth%)*3)'          ! und unter-
        Print #99,String$(Len(Pfad$+File$)+10,"=")! streichen
        Inc Dx_level.999%       ! Ebenenzähler +1
        Getdir(Path1$+"\"+File$+Path2$,Attr%,Fname$) ! Nächste
                                ! Ebene durchsuchen
        Dec Dx_level.999%       ! Zähler wieder zurücksetzen
        Astring(0,Pfad$,"\",Depth$) !--.  alten
        ' Die Routine Astring          |- Ebenenindex
        ' finden Sie unter INSTR       |  ermitteln
        Depth%=Len(Depth$)/2        !--'
        X%=Fsfirst(Pfad$,Attr%)! Suchpfad restaurieren
        Goto Sta                ! Rücksprung zur höheren Ebene
      Else                      ! Datei ist kein Ordner!
        If (D.att¦ And 8)       ! Disk-Label?
          Print #99,"'";Spc((Depth%-1)*3);" LABEL : ";File$;
        Else                    ! Normaler Dateiname!
          Sek¦=(D.uhr& And 31)*2          !---------.
          Min¦=(D.uhr& Div 32) And 63              !  Dateizeit
          Std¦=(D.uhr& Div 2048) And 31            !- und Datum
          Tag¦=D.dat& And 31                       !  ermitteln
          Mon¦=(D.dat& Div 32) And 31              !
          Jhr&=((D.dat& Div 512) And 31)+1980 !--'
          Td$=Right$("0"+Str$(Std¦),2)+":"    !--.
          Td$=Td$+Right$("0"+Str$(Min¦),2)+":"  !  Ausgabe-
          Td$=Td$+Right$("0"+Str$(Sek¦),2)+"  " !- String
          Td$=Td$+Right$("0"+Str$(Tag¦),2)+"."  !  bilden
          Td$=Td$+Right$("0"+Str$(Mon¦),2)      !
```

```
              Td$=Td$+"."+Str$(Jhr&) !--------------'
              Print #99,"'";Spc((Depth%-1)*3);" DATEI : "; ! Datei-
              Print #99,File$;Spc(12-Len(File$)); ! Namen ausgeben
              Print #99,Using "##  & #######",D.att|,Td$,D.len%;
              '                       ! Daten formatiert ausgeben
            Endif
          Endif
          Print #99                   ! CR/LF schreiben
        Endif
      Endif
      X%=Fsnext()                     ! Mehr Files auf dieser Ebene?
    Wend                              ! Dann zum nächsten File-Cheque
    Print #99,"'"                     ! Leerzeile als Trennung
    If Dx_level.999%=0                ! Wieder auf oberster Ebene?
      Close #99                       ! Dann Ausgabekanal schließen
    Endif
  Return
```

Die "Noch-V2.xx-Besitzer" sollen hier nicht benachteiligt werden. Auf der Diskette im Buch (siehe auch Kapitel 1.1) finden Sie ein Programm DISK_DIR.LST welches generell denselben Aufbau hat wie das hier gezeigte V3.0-Listing. Der Unterschied besteht natürlich darin, daß die Nur-V3.0-Befehle und -Variablentypen simuliert bzw. ersetzt wurden.

KILL { K/V3.0: KI } — Disk-Datei löschen

KILL "Dateiname"

Möchten Sie eine Datei auf dem Festspeicher (Diskette/Hard-Disk/RAM-Disk) löschen, können Sie das mit diesem Befehl tun. Dazu wird in "Dateiname" der Name der zu löschenden Datei (Pfad siehe ggfs. Kapitelanfang) angegeben.

LIST { LIS } — Programm listen/speichern (ASCII)

LIST ["Dateiname"]

Möchten Sie das im Arbeitspeicher befindliche Programm auf dem Monitor auflisten oder als ASCII-File auf Diskette abspeichern, hilft Ihnen dieser Befehl. Geben Sie im Direktmodus oder als Programmzeile nur den Befehlsnamen ohne Angabe eines Dateinamens ein, wird das gesamte Programm auf dem Ausgabebildschirm ausgegeben. Das Listing kann jederzeit durch die GFA-Abbruchtastenkombination <Control><Shift><Alternate> unterbrochen werden.

Wollen Sie das Listing als sogenanntes ASCII-File auf Diskette abspeichern, übergeben Sie dem Befehl (in Anführungsstrichen) den

Namen der Datei, in welcher das Listing abgelegt werden soll. Dieser Befehl ist identisch mit der Editor-Funktion Save,A. Sie können also hiermit abgespeicherte Programme jederzeit mit der Editor-Funktion Merge in ein anderes, im Arbeitsspeicher befindliches Programm einbinden. Die Anführungsstriche zum Dateinamen können vernachlässigt werden, da sie vom Interpreter - falls nicht vorhanden - selbsttätig gesetzt werden. Die Extension zum Dateinamen kann ebenfalls vernachlässigt werden. Auch sie wird vom Interpreter gesetzt (.LST - falls nicht anders vorgegeben). Befindet sich bereits ein Programm gleichen Namens auf der Diskette, wird es automatisch auf Programmname.BAK umbenannt.

Ein ASCII-File ist eine Datei, in welche jedes einzelne Text- und Steuerzeichen des zu speichernden Textes in der Reihenfolge seines Auftretens als entsprechender ASCII-Wert (0 - 255) geschrieben wird. Die meisten textverarbeitenden Programme besitzen die Möglichkeit, ASCII-Files zu speichern und zu laden. D.h., daß Texte dieser Art unter den verschiedenen Programmen, Systemen und Computern austauschbar sind, soweit diese sich an den American Standard Code for Information Interchange halten (ASCII = engl.: Amerikanischer Standard-Code zur Informationsübertragung). Dateien mit anderen Formaten, wie z.B. die .BAS-Dateien (bzw. .GFA in V3.0) werden nach einem anderen Schema codiert. Dieser Token-Code wird vom Programmierer selbst entwickelt, um zum Beispiel eine höhere Lade- oder Speichergeschwindigkeit, einen speziellen Effekt oder eine erschwerte Lesbarkeit zu erreichen (siehe PSAVE). Diese Dateien sind dann allerdings nicht mehr mit anderen Programmen (Interpretern etc.) austauschbar. Sie sind nicht mehr kompatibel (compatibility = engl.: Vereinbarkeit/Verträglichkeit).

Zur Ausgabe des Listings auf dem Drucker haben Sie auch die Möglichkeit, in Dateiname den Drucker-Port PRN: anzugeben. Diese Variante ist dann identisch mit dem Befehl LLIST, wobei jedoch Punkt-Befehle (siehe Editor-Funktion Llist) unberücksichtigt bleiben.

Es kann manchmal sehr nützlich sein, mehr Platz auf einer Diskette schaffen zu können, ohne dabei auf die wichtigen Daten verzichten zu müssen. In den meisten Fällen ist dies nicht möglich, da die Struktur der Dateien und die Art und Weise des Zugriffs aus einem Programm heraus nicht bekannt sind.

Bei GFA-.LST-Files, also reinen ASCII-Dateien, ist das Format und die Art und Weise des Zugriffs durch den Interpreter bekannt. Aus diesem Grund können wir es uns leisten, diese Dateien im Rahmen

der Möglichkeiten zu verändern. Mit dem folgenden Kompressor-Programm ist das ganz einfach.

Beim Einlesen von ASCII-Dateien durch die Merge-Funktion überprüft der Interpreter die Richtigkeit der gelesenen Zeilen. Wenn man nun weiß, daß die Programmstruktur vom Interpreter immer erst dann aufgebaut wird, wenn die entsprechenden Programmteile im Editor angezeigt werden und diese optische Struktur für den Interpreter selbst eigentlich unwichtig ist, ist der Gedanke, daß man auf diese Struktur auch in den LST.-Files verzichten kann, garnicht so weit weg. In der ASCII-Datei wird die optische Struktur einfach durch entsprechend viele Space-Zeichen (Chr$(32)) gebildet. Um nun das Disketten-File zu kürzen, brauchen wir als erstes also nur diese Spaces herauszuoperieren.Zweitens kann an jedem Programmzeilen-Ende auf das Line-Feed-Zeichen (Chr$(10)) verzichtet werden. Dem Interpreter genügt zum Erkennen des Zeilenendes das Carriage-Return-Zeichen (Chr$(13)). Das allein kann bei umfangreichen Programmen mit mehreren tausend Zeilen schon einige KByte ausmachen.

Die dritte Möglichkeit, Platz einzusparen, besteht darin, alle Befehlsnamen, die auch durch Abkürzungen eingegeben werden können, durch diese Abkürzungen zu ersetzen. Bei diesen Befehlsnamen handelt es sich grundsätzlich um Befehle, die nur direkt am Zeilenanfang auftreten können. Funktionen - die auch innerhalb einer Zeile stehen können - verfügen über keine Abkürzungen. Der Interpreter erkennt beim Einlesen des .LST-Files diese Abkürzungen. Ein GFA-BASIC-Programm wird im Speicher generell als Mnemonic-Code (bzw. Token-Code) abgelegt und nur bei der Bildschirm-Anzeige des Listings im Editor, bzw. beim TRON- und LIST-Befehl in das für den Programmierer erkennbare Listing umgewandelt.

Dieses Programm vollzieht nun diese drei Schritte und schreibt die dadurch komprimierte Datei wieder auf die Diskette. Im Allgemeinen sind damit Platzeinsparungen von bis zu 30 Prozent möglich. Ein .LST-File von 100 KByte Länge hätte demnach hinterher eine Länge von nur 70-75 KByte.

Aus programmtechnischer Sicht ist das besondere an diesem Programm, daß es nicht über ein String-Feld die Zeile speichert und bearbeitet, sondern das gesamte Programm als Block in das freie RAM lädt. Anschließend werden die Zeile einzeln in einen String-Puffer gelesen und dort auf kleinstmögliches Format getrimmt. Ist das geschehen, wird die gekürzte Zeile wieder ins freie RAM transportiert.

Der Kompressor ist nicht mehr jung. Ich habe ihn vor ca. eineinhalb Jahren geschrieben. Ich habe ihn für dieses Buch nicht effektiv verändert, da ich glaube, daß er wohl am besten dazu geeignet ist, als Übungsobjekt für jegliche Art der Optimierung oder Erweiterung zu dienen. Wenn Sie also Spaß daran haben, versuchen Sie doch einfach, dieses Programm in die V3.0-Version zu übertragen und anzupassen.

```
On break gosub Schluss      ! Abbruch abfangen
Deffill ,2,4
Pbox 10,10,629,389
Alert 2,">.LST-FILE-COMPRESSOR<",3,"DO IT",Flag%
Al$="Wurde das LST-File mit|Deflist 0 (BEFEHLSNAME) oder|"
Al$=Al$+"Deflist 1 (Befehlsname)|abgespeichert?"
Alert 2,Al$,1,"Deflist1|DEFLIST0",D.efflag%
Restore Befehle             ! DATA-Zeiger setzen
Do                          ! Erste Lese-Schleife
  Read Dumm$                ! DATA lesen
  Exit if Instr(Dumm$,"XXX") ! Exit, wenn Endmarkierung erreicht
  Inc D.atas%               ! Mitzählen
Loop
Div D.atas%,2               ! DATA-Anzahl/zwei, da immer zwei
'                           ! DATAs pro Befehl
Restore Befehle             ! Noch einmal DATA-Zeiger setzen
Dim Lang$(D.atas%),Kurz$(D.atas%) ! Befehls-Array dimensionieren
For I%=1 To D.atas%         ! Alle Befehle und Abkürzungen...
  Read Lang$(I%),Kurz$(I%)  ! ...lesen
  If Defflag%=2             ! Mit DEFLIST 0 abgespeichert?
    Lang$(I%)=Upper$(Lang$(I%)) ! Dann Befehl in Großbuchstaben
  Endif
Next I%                     ! Nächster Befehl
Reserve 30000               ! 30 KByte Reserve für BASIC
Fileselect "\*.lst","",S$   ! Zu komprimierende Datei wählen
If Exist(S$)                ! Datei existiert?
  Open "I",#1,S$            ! Datei öffnen
  Size%=Lof(#1)             ! Dateilänge feststellen
  Bload S$,Himem            ! Datei hinter BASIC-Speicher laden
  Print At(10,10);"Comprimiere Zeile : "
  Do                        ! Kompressor-Schleife
    Count%=0                ! Byte-Zähler auf Null
    A$=""                   ! Zeilenpuffer löschen
    Do                      ! Leseschleife
      Byte%=Peek(Himem+Offset%+Count%) ! Einzelzeichen selektieren
      Inc Count%            ! Byte-Zähler +1
      A$=A$+Chr$(Byte%)     ! Zeile zusammenfügen
      Exit if Byte%=13      ! Exit, wenn Carriage Return
      ' Als kleine Hausaufgabe wäre es hier interessant, die
      ' Cchar-Prozedur (siehe CHAR()) einzubinden.
    Loop
    Add Offset%,Len(A$)+2   ! Positions-Zähler addieren
    '                       ! (+2 bedeutet, daß die Zeichen
    '                       ! CR und LF mitgezählt werden müssen)
    Exit if Offset%>Size%   ! Exit, wenn Ende des alten Listings
    Inc D.one%              ! Zeilen mitzählen
    Print At(43,10);D.one%  ! Aktuelle Zeile ausgeben
    While Left$(A$)=" "     ! Zeilenanfang = Space?
      A$=Right$(A$,Len(A$-1)) ! Dann Space löschen
    Wend                    ! Nächstes Zeichen
```

```
    For I%=1 To D.atas%     ! Befehlsliste durchgehen
      If Left$(A$,Len(Lang$(I%)))=Lang$(I%) ! Listenwort gefunden?
        A$=Right$(A$,Len(A$)-Len(Lang$(I%)))! Restzeile rechts...
        '                   !          ...vom Befehlsnamen bilden
        A$=Kurz$(I%)+A$     ! Abkürzung einsetzen
      Endif
    Next I%                 ! Nächster Befehl der Liste
    Bmove Varptr(A$),Himem+Size%+Gesamt%,Len(A$)
    '                       ! Zeile komprimiert zurück
    Add Gesamt%,Len(A$)     ! Bisher komprimierte Länge
    Exit if E.flag%=1       ! Ende, wenn Exit-Flag gesetzt
  Loop                      ! Große Schleifen-Wende
  If E.flag%=1              ! Exit-Flag gesetzt?
    Goto Raus               ! Restprogramm überspringen
  Endif
  Startkill:
  Alert 2,"Altes LST-File löschen?",1,"OKAY|NEIN",Back%
  If Back%=1                ! Altes .LST-File löschen?
    If Exist(S$)            ! Diskette nicht gewechselt?
      K.flag%=1
      Kill S$               ! Datei löschen
    Else                    ! Diskette gewechselt!
      Al$="File nicht gefunden !|Diskette wurde gewechselt !"
      Alert 1,Al$,1,"Nochmal|Weiter",Back%
      If Back%=1            ! Noch einmal versuchen?
        Goto Startkill      ! Dann zurück
      Endif
    Endif
  Endif
  Startsave:
  If Dfree(0)>Gesamt%+1000 ! Reicht Diskettenplatz aus?
    F.name$=Right$(S$,12)  ! Dateinamen selektieren
    Backslash%=Instr(F.name$,"\") ! Backslash enthalten?
    F.name$=Right$(F.name$,Len(F.name$)-Backslash%)
    '                       ! Rest-String = Dateiname
    If K.flag%=0            ! Altes File nicht gelöscht?
      F.name$=Left$(F.name$,Len(F.name$)-4)+".BAC" ! Ext. .BAC setzen
    Endif                   ! Extension .BAC setzen
    Fileselect "\*.*",F.name$,F.name$
    If F.name$<>"" And F.name$<>"\" ! Gültiger Dateiname?
      Bsave F.name$,Himem+Size%,Gesamt% ! Block auf Disk zurück
      Inc C.opy%            ! Kopien mitzählen
      Alert 2,Str$(C.opy%)+". KOPIE ablegen??",1,"Nein|Okay",Back%
      If Back%=2            ! Noch eine Kopie?
        Goto Startsave      ! Dann noch einmal speichern
      Endif
    Endif
  Else                      ! Diskettenplatz reicht nicht
    Al$="Disketten-Speicherplatz reicht|nicht aus! Evtl. Disk-Wechsel!"
    Alert 1,Al$,1,"Nochmal|Abbruch",Back%
    If Back%=1              ! Noch einmal versuchen?
      Goto Startsave        ! Dann noch einmal
    Endif
  Endif
Endif
Raus:
Reserve Xbios(2)-Himem-16384+Fre(0) ! BASIC-Speicher restaurieren
'
```

```
Procedure Schluss
  '
  ' Diese Break-Abfang-Routine ist notwendig, um bei Programmende
  ' den BASIC-Speicher wieder ordnungsgemäß restaurieren zu können.
  '
  E.flag%=1                  ! Abbruch-Flag setzen
  Cont                       ! Weiter geht's
Return
```

Die folgende DATA-Liste ist noch nicht vollständig. Sollten Ihnen - gerade zu V3.0 - noch weitere Abkürzungen einfallen, brauchen Sie diese einfach nur in diese Liste eintragen. Für die V3.0-Version beachten Sie bitte, daß sich einige Befehlsabkürzungen von V2.xx-Befehlen in V3.0 geändert haben (z.B. TITLEW).Bei der Erweiterung der DATA-Liste ist außerdem zu beachten, daß der ausgeschriebene und der abgekürzte Befehlsname immer als Paar (erst ausgeschrieben, dann abgekürzt) direkt hintereinanderstehen und daß als letztes DATA-Element der String "XXXXX" verwendet wird.

Befehle:

```
Data Pbox,pb,Return,ret,Repeat,rep
Data Procedure,pro,Print,p,Box,b,Until,u
Data Loop,l,For ,f ,Next,n,Fill,fi,Mouse,m
Data Put,pu,Text,t,Defline,de,Swap,sw
Data Line,li,Plot,pl,Local,loc,Draw,dr
Data Data,d,Defmark,defm,Defmouse,defmo
Data Deffill,deff,Deftext,deft,Dim,di
Data Erase,er,Endif,endi,Alert,a,Input,inp
Data Restore,res,Circle,c,Color,co
Data Setcolor,se,Pcircle,pc,Ellipse,ell
Data Pellipse,pe,Prbox,prb,Rbox,rb,Else,el
Data Poke,po,Lpoke,lp,Pause,Pa,Goto,got
Data Resume,resu,Read,rea,If,i,Inc,in
Data Add,ad,Sub,s,Rem,',Gosub,@
Data Reserve,rese,Bmove,bm,Showm,sh
Data Hidem,hi,Option Base,opt base
Data Option ",opt ",Void,vo,Vdisys,V
Data Void,vo,Gemsys,ge,Open ,o ,Edit,ed
Data Fileselect,filese,Graphmode,g,Bload,bl
Data Bsave,bs,Dpoke,dp,Monitor,mon,XXXXX
```

LOAD { LOA } — Programm in Abeitsspeicher laden

LOAD "Programmname"

Mit LOAD kann ein beliebiges GFA-BASIC-Programm in den Arbeitsspeicher geladen werden. Prinzipiell verhält sich dieser Befehl so, als würden Sie den Menüpunkt Load im Editormenü anklicken oder im Editor die Taste <F1> drücken. Auch hier ist die am Kapitelanfang beschriebene Pfadstruktur zu verwenden. "Programmname" ist der

Name des zu ladenden Programms. Wird dem Programmnamen keine Extension beigefügt, wird vom Interpreter selbstständig .BAS angehängt, z.B.:

```
Load "\Utility\Drei_d"
```

Das aktuelle Programm wird beendet und muß durch Run im Direktmodus, <Shift><F10> oder Klick auf Run im Editormenü gestartet werden. Bei durch PSAVE gespeicherten Programmen ist dies jedoch nicht nötig, da sie selbsttätig gestartet werden. Soll das Programm compiliert werden, wird LOAD vom Compiler nach Abfrage ggfs. ignoriert.

MKDIR {MK} — Ordner erzeugen

MKDIR "Ordner"

Mit MKDIR können auf einer beliebigen Station im Haupt-Directory oder innerhalb eines vorhandenen Ordners weitere Ordner angelegt werden. Dabei wird ggfs. wieder die am Kapitelanfang beschriebene Pfadstruktur verwendet. Als Überblick folgt eine Auflistung verschiedener Varianten dieses Befehls.

Mkdir "akte"y

Erzeugt einen neuen Ordner mit dem Namen AKTE auf der aktuellen Station entweder im Haupt-Directory bzw. in dem Ordner, der gerade geöffnet ist.

Mkdir "B:\Fach_a\Akte"

Erzeugt einen neuen Ordner mit Namen AKTE auf Station B im vorhandenen Ordner FACH_A

A$="\fach_a\"
Mkdir A$+"akte"

Erzeugt einen neuen Ordner mit Namen AKTE auf der aktuellen Station im schon vorhandenen Ordner FACH_A.

Ist bereits ein Ordner mit derselben Pfadbezeichnung vorhanden, wird eine Fehlermeldung ausgegeben. Die Pfadbezeichnung und der Ordnername wurden hier mit Absicht klein geschrieben, um deutlich zu machen, daß das GFA-BASIC diese Angaben selbstständig in

Großbuchstaben umwandelt. Dies ist grundsätzlich bei allen Diskettenoperationen der Fall. Zurückgegeben werden die Dateinamen vom System allerdings ausschließlich in Großbuchstaben (siehe Rückgabe-String bei FILESELECT oder DIR$).

NAME { NA AS/V3.0: NA } Datei umbenennen

NAME "Name_alt" AS "Name_neu"

Ermöglicht die nachträgliche Änderung eines Dateinamens. Dieser Befehl erwartet zwei Parameter. Der erste davon (Name_alt) ist der Dateiname, dessen Name verändert werden soll. nach einem angehängten AS wird ein weiterer Name (Name_neu) übergeben, der nun an die Stelle des alten Namens tritt. Beide Namen können entweder als String-Ausdruck, als String-Variable oder als Kombination aus beiden übergeben werden. Am Kapitelanfang finden Sie die Pfadstruktur, die ggfs. auch hier zu verwenden ist. Zu beachten ist bei diesem Befehl, daß eine evtl. Stationsvorgabe (z.B. A:\) bei beiden Namen identisch anzugeben ist. Ist beim alten Namen eine Diskettenstation angegeben und ist diese Station auch die aktuelle, dann kann beim neuen Namen die Spezifikation entfallen. Innerhalb einer Station kann auch ohne weiteres eine im Haupt-Directory verzeichnete Datei durch eine entsprechende Pfadangabe im neuen Namen in einen schon existierenden Ordner verlegt werden.

z.B.

```
Mkdir "Akte"                    ! Ordner anlegen
Fileselect "\*.*","",Aa$        ! Datei wählen
If Exist(Aa$)                   ! Datei existiert?
  Name Aa$ As "\Akte\"+Aa$      ! Dann umbenennen
Endif
```

In Version V3.0 kann statt des Befehlsteils AS auch ein Komma oder Leerzeichen verwendet werden. An- und Ausführungszeichen sind optional.

PSAVE { PS/in V3.0: PSA } Programm speichern (listgeschützt)

PSAVE "Programmname"

Für diesen Befehl gilt die gleiche Ausführung wie zu SAVE. Er bietet nur eine kleine Besonderheit, die aber in ihrer Wirkung sehr beeindruckend ist. Das P in diesem Befehlsnamen namen steht für protected

(engl.: geschützt). Die BAS-Programme, die hiermit abgespeichert wurden...

- Können nicht mehr gelistet werden.
- Werden sofort nach dem Laden gestartet (Autostart, s. LOAD).
- Können nicht bearbeitet werden. Wird versucht, mit dem Interpreter das - unsichtbare - Listing zu bearbeiten, wird man früher oder später mit einem Total-Absturz belohnt. Der Interpreter verfügt bei PSAVE-Programmen nicht mehr über die notwendigen Zeiger auf die Variablennamen und beim Versuch, diese zu finden gibt es Adressen-Wirrwarr.

Version 3.0

RENAME { REN } — Datei umbenennen

RENAME "Name_alt" AS "Name_neu"

Entspricht exakt dem Befehl NAME (weiteres siehe dort).

RMDIR { RM } — Ordner löschen

RMDIR "Ordner"

Die Syntax dieses Befehls ist identisch mit CHDIR, nur daß der angegebene Ordner nicht geöffnet, sondern gelöscht wird. Befinden sich allerdings noch Dateien in dem Ordner, kann dieser Befehl nicht ausgeführt werden und es erscheint eine Fehlermeldung. Ggfs. sind die enthaltenen Dateien vorher durch KILL zu löschen.

SAVE { SA } — Programm speichern (codiert)

SAVE "Programmname"

SAVE speichert das im Programmspeicher befindliche Programm unter dem angegebenen Namen im Token-Format auf dem Festspeicher (Diskette/Hard-Disk) bzw. RAM-Disk (zur Pfadangabe siehe ggfs. Kapitelanfang). Die Anführunsstriche zum Dateinamen können vernachlässigt werden, da sie vom Interpreter - falls nicht vorhanden - selbsttätig gesetzt werden. Die Extension zum Dateinamen kann ebenfalls vernachlässigt werden. Auch sie wird vom Interpreter gesetzt (V2.xx: .BAS/V3.0: .GFA - falls in Programmname keine andere Extension vorgegeben wurde). Befindet sich bereits ein Programm glei-

chen Namens auf der Station, wird es automatisch auf Programmname.BAK umbenannt.

5.5 Dateihandhabung

BGET { BG } Teildatei lesen

BGET [#]Kanal,Start,Anz

Kanal ist der Identifikator einer mit OPEN "I" oder "U" geöffneten Datei. Ab File-Pointer-Position werden Anz Bytes dieser Datei in den Arbeitsspeicher - beginnend mit der Adresse Start - gelesen. Wird die Datei anschließend nicht mit CLOSE geschlossen, bleibt der File-Pointer auf der dem gelesenen Teil folgenden Byte-Position stehen. Der nächste Dateizugriff (BGET#, INP(#), INPUT#, PRINT#, OUT# etc.) bezieht sich dann auf diese Position.

Beispiel:

```
Open "O",#1,"Test.Dat"     ! Output-Datei öffnen
Print #1;"GFA-BASIC";      ! String hineinschreiben
Close                      ! und wieder schließen
A$=Space$(5)               ! Kleinen Puffer setzen
Open "I",#1,"Test.Dat"     ! Datei für Eingabe öffnen
Seek #1,4                  ! 4 Bytes überspringen
Bget #1,Varptr(A$),5       ! 5 Bytes in Puffer lesen
Close                      ! Datei schließen
Print A$                   ! String ausgeben
```

Weitere Anwendungsmöglichkeit s. BPUT bzw. RAMKART-Listing/Procedure Datload.

BPUT { BP } Teildatei schreiben

BPUT [#]Kanal,Start,Anz

Es werden Anz Bytes ab der Adresse Start gelesen und - beginnend mit der File-Pointer-Position - in eine mit OPEN "O" bzw. "U" geöffnete Datei mit dem Identifikator Kanal geschrieben. Im Gegensatz zu BSAVE können hiermit auch Teile einer Datei überschrieben oder an diese angefügt werden. Wird die Datei anschließend nicht mit CLOSE geschlossen, bleibt der File-Pointer auf der dem geschriebenen Teil folgenden Byte-Position stehen. Der nächste Dateizugriff (BGET#, INP(#), INPUT#, PRINT#, OUT# etc.) bezieht sich dann auf diese Position.

Beispiel:

```
Yt%=Min(2,3-Xbios(4))     ! Y-Auflösungsteiler
Open "O",#1,"test"        ! File für Schreiben öffnen
Print #1,Mki$(25)+Mki$(161)+Mki$(1); ! Eigenen PUT-Header
'                         ! voranstellen (ohne CR/LF!)
For I%=0 To 5             ! 6 Icons >----------------------.
  Deffill ,2,I%+1         ! DEFFILL Graustufen             |
  Pbox 100+I%*40,200/Yt%,100+I%*40+25,225/Yt%   ! Zeichnen |
  Get 100+I%*40,200/Yt%,100+I%*40+25,226/Yt%,A$ ! 'GET'ten |
  Bput #1,Varptr(A$)+6,Len(A$)-6 ! Ohne Header speichern   |
Next I%                   ! Nächstes Icon <----------------'
Close #1                  ! File schließen
Open "I",#1,"test"        ! File für Lesen öffnen
Buff$=Space$(Lof(#1))     ! Puffer vorbereiten
Bget #1,Varptr(Buff$),Lof(#1) ! File in Puffer lesen
Close #1                  ! File schließen
For I%=0 To 5             ! Fünf mal >---------------------.
  Put 100+I%*40,(200-I%*27)/Yt%,Buff$ ! neues Icon setzen  |
  Pause 20                ! Kleine Pause                   |
Next I%                   ! Nächste <----------------------'
```

Weitere Anwendungsmöglichkeit siehe RAMKART-Listing/Procedure Update.

CLOSE { CL } — Datenkanal schließen

CLOSE [#Kanal]

Alle mit OPEN eröffneten Kanäle müssen mit CLOSE #Kanal geschlossen werden, um ihre Kanal-Nummer für eine andere Datei verwenden zu können. Wird CLOSE ohne #Kanal verwendet, werden damit sämtliche Kanäle gleichzeitig geschlossen. Ein Kanal der noch nicht mit CLOSE #Kanal geschlossen wurde und nochmals mit OPEN angesprochen wird, verursacht eine Fehlermeldung. Bei Compilaten werden bei Programmende alle noch offenen Dateien automatisch geschlossen. Im Interpreterbetrieb können nach Programmende von der Direkt-Ebene aus alle offenen Dateien weiterhin angesprochen werden, solange im Editor keine Programmänderung vorgenommen wurde.

EOF() — Datei auf Dateiende prüfen

Var=EOF(#Kanal)

Mit dieser Funktion kann - außer bei virtuellen Dateien (AUX:, VID: etc.) - festgestellt werden, ob sich der File-Pointer hinter dem letzten Byte der mit #Kanal angegebenen Datei - also am Dateiende (EndOf-File) - befindet. Ist dies der Fall erhält man den Wert -1 (TRUE), an-

dernfalls eine Null (FALSE). Die wohl wichtigste Anwendung dieser Funktion ist die Vermeidung der Fehlermeldung "File-Ende erreicht".

Beispiel:

```
Open "I",#1,"Dateiname"   ! File zum Lesen öffnen
While Eof(#1)=False       ! Solange File-Ende nicht erreicht <--.
  Out 5,Inp(#1)           ! Einzelnes Byte lesen und ausgeben   |
Wend                      ! Wieder von vorn >-------------------'
```

FIELD { FI AS bzw. FI AT } Datensatz in Felder unterteilen

FIELD #Kanal,Anz AS Var1$ [,Anz AS Var2$,...]
FIELD #Kanal,Anz AT(Adr1) [,Anz AS Var$,...] **(nur V3.0)**

Teilt die Datensätze der mit Kanal# angegebenen Random-Datei in soviele Einträge auf, wie durch die Menge der Anz AS Var$-Komponenten vorgegeben wird. Es wird jeweils die in Anz angegebene Anzahl an Bytes der nach AS folgenden String-Variablen (Var1$, Var2$ etc.) zugeordnet und die Variable gleichzeitig mit Leerzeichen gefüllt.

In V2.xx ist zu beachten, daß für jeden geöffneten Datenkanal im R-Modus nur eine FIELD-Anweisung zulässig ist. Wird versucht, mehrere FIELD-Anweisungen an denselben Datenkanal auszugeben, erscheint die Fehlermeldung "Nur ein Field zu einem Open "R" möglich". Es ist also in V2.xx bei größeren Datenfeldern ratsam, die Namen der aufnehmenden String-Variablen so kurz wie möglich zu wählen, da FIELD hier in seiner Länge durch die maximale Eingabezeilenlänge von 255 Zeichen begrenzt wird.

Um numerische Daten nicht durch MKI$ etc. in Strings umwandeln zu müssen, kann in V3.0 die zweite Syntax-Variante verwendet werden. Dabei enthält Anz die Anzahl an Bytes, die ab der zugehörigen - hinter AT in Klammern gesetzten - Adresse gelesen werden sollen.

Beispiel:

```
a%=673123
b%=VARPTR(a%)
c&=1000
d=61234.1231
FIELD #1,4 AT(b%),2 AT(*c&),8 AT(*d)
```

AT- und AS-Komponenten können in einer FIELD-Zeile beliebige gemischt werden (z.B. FIELD #1,20 AS a$,2 AT(*b&),...). Außerdem kann in V3.0 die FIELD-Aufteilung für eine Datei auf mehrere Pro-

grammzeilen verteilt sein. Diese werden dann als zusammengehörig angenommen, was die oben erwähnte Fehlermeldung überflüssig macht. Außerdem kann in V3.0 statt des Befehlsteils AS auch ein Komma oder Leerzeichen angegeben werden.

Siehe weiteres Beispiel unter 5.5.1 "Funktionsweise einer Random-Access-Datei".

GET # — Datensatz lesen

GET #Kanal [,Satznummer]

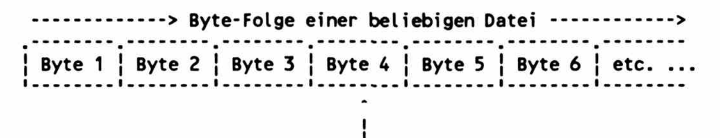
Kanal gibt die R-Datei (siehe OPEN) an, aus welcher der in Satznummer angegebene Datensatz gelesen werden soll. Bei der Zuordnung von Datensätzen zu einer R-Datei ist jedem Satz eine Nummer zuzuteilen. Durch deren Angabe kann der entsprechende Datensatz mit GET# wieder in die mit FIELD spezifizierten String-Variable(n) zurückgelesen werden. Fehlt "Satznummer", wird jeweils der nächste - bzw. in V3.0 der durch RECORD ggfs. gesetzte - Datensatz gelesen. Beispiel unter 5.5.1 "Funktionsweise einer Random-Access-Datei".

LOC() — File-Pointer-Position

Var=LOC(#Kanal)

LOC (Abk.f. Location) liefert die aktuelle Position des Schreib- und Lesezeigers (File-Pointer) der durch #Kanal bestimmten Datei. Die gelieferte Byte-Position wird ab Dateianfang gezählt.

Beispiel:



Angenommen: Aktueller Pointer zeigt auf Byte 4:

```
Pointer%=Loc(#1)    ! Liefert dann in Pointer% den Wert 4
```

LOF() Dateilänge ermitteln

Var=LOF(#Kanal)

Durch LOF (Abk.f. Length Of File) ist es bei Disketten- bzw. Hard-Disk-Dateien möglich - sofern Sie überhaupt einen Inhalt haben - ihre Byte-Länge zu ermitteln. Der Funktion wird dazu in Klammern die #Kanal-Nummer der betreffenden Datei übergeben.

Beispiel:

```
Open "U",#1,"Dateiname"  ! Zuerst Datei öffnen
Print Lof(#1)            ! Variante 1: Direktausgabe
A%=Lof(#1)               ! Variante 2: Zuweisung
If Lof(#1)>32767         ! Variante 3: Bedingungsabfrage
  A$=Input$(32767,#1)
Else
  A$=Input$(Lof(#1),#1)  ! Variante 4: Einbindung
Endif
... etc.
```

Weitere Anwendungsmöglichkeit s. BPUT bzw. RAMKART-Listing/ Procedure Datload.

OPEN {O} Datenkanal öffnen

OPEN "Modus",#Kanal,"Dateiname" [,Satzlänge]

Dieser Befehl eröffnet eine Datei - und Ihnen damit die bunte Welt der Dateiverwaltung. Seine Syntax ist auf den ersten Blick etwas kompliziert. Nach näherem Kennenlernen gewinnt man ihn jedoch schnell lieb. Die Möglichkeiten, die sich Ihnen mit den Befehlen zur Dateiverwaltung bieten, sind schier unüberschaubar.

Modus:

"O"	Output öffnet eine Datei bzw. installiert sie neu. Existiert die angegebene Datei (auf Disk bzw. Hard-Disk) bereits, wird deren Länge auf Null gesetzt. In diesem Modus sind nur Schreibzugriffe zulässig.
"I"	Input öffnet eine vorhandene Datei zum Herauslesen von Daten. Der File-Pointer wird auf das erste Byte der Datei gesetzt. Die Dateilänge bleibt durch Lesezugriffe unverändert. In diesem Modus sind nur Lesezugriffe zulässig.
"A"	Append öffnet eine vorhandene Datei und setzt den File-Pointer auf das Dateiende. Alle an diese Datei auszugebenden Daten werden an das Dateiende angehängt (append = engl.: anhängen/hinzufügen).
"U"	Update öffnet eine Datei für gleichzeitigen Schreib- und Lesezugriff. Dabei bleibt die ursprüngliche Länge erhalten. Es sei denn, bei Datenausgabe wird über das aktuelle Dateiende hinaus geschrieben.

"R" Random öffnet eine Random-Access-Datei. Maximal sind 65535 Datensätze pro Datei möglich. Genauere Informationen hierzu finden Sie unter Kapitel 5.5.1. "Funktionsweise einer Random-Access-Datei".

#Kanal:

Es können gleichzeitig max. 100 Dateien eröffnet sein. Bei der Eröffnung wird zur Identifikation jeder Datei eine Zahl zugewiesen. Unter Angabe dieser Zahl hinter dem Nummernzeichen kann nun z.B. durch Befehle wie PRINT#, INPUT#, BGET# etc. auf jede einzelne dieser Dateien zugegriffen werden. Die angegebene #Kanal-Nummer muß im Bereich von 0 bis 99 liegen.

Dateiname:

Bezeichnet den Namen der Datei, die mit dem betreffenden Zugriffsmodus geöffnet werden soll. Hier kann wieder die schon erwähnte Pfadstruktur (siehe Kapitelanfang) verwendet werden.

Ausnahme:

Sie können mit fast allen #-Dateibefehlen auch die verschiedenen Schnittstellen Ihres Atari ST ansprechen. Dazu wird ebenfalls eine Quasi-Datei (virtuelle Datei) geöffnet. Mit Datei ist in diesem Fall jedoch nicht eine Disketten- bzw. Hard-Disk-Datei gemeint, sondern der jeweilige Port.

Es stehen Ihnen sechs solcher Ports zur Auswahl:

CON: = Consol-Port
LST: = Seriell-Port (Drucker-Port)
PRN: = Drucker-Port (Seriell-Port)
AUX: = Auxilliary-Port
MID: = MIDI-In/Out-Ports
VID: = Monitor (CON: transparent)
IKB: = Tastaturprozessor

Der Ausdruck vor dem Gleichheitszeichen (inkl. Doppelpunkt) wird in diesem Fall als Dateiname an OPEN übergeben. Die Angabe von Modus kann bei Verwendung dieser Schnittstellen-Variante entfallen.

Beispiel:

```
Open "",#1,"VID:"           ! VIDEO-Port öffnen
For I%=0 To 31              ! 32...  >-----------------.
   Print #1;Chr$(I%);       ! ...Sonderzeichen ausgeben |
Next I%                     ! Nächstes Zeichen <--------'
Close #1                    ! Port wieder schließen
```

Dieses Beispiel demonstriert eine weitere Möglichkeit (siehe auch OUT), ohne Tastaturbedienung die Sonderzeichen mit den ASCII-Werten von 0 bis 31 zu Gesicht zu bekommen.

Satzlänge:

Mit Random-Access-Dateien ist es möglich, sich eine Reihe von Datensätzen innerhalb einer Datei anzulegen, die dann durch Identifikatoren angesprochen werden können. Der OPEN-Befehl erweitert sich ggfs. hierfür um den Parameter Satzlänge. Darunter ist ein Wert zu verstehen, den Sie als Dateiverwalter selbst bestimmen können. Und zwar gibt er die Anzahl der Bytes an, die sie den Datensätzen einer Datei zukommen lassen wollen. Wird diese Angabe bei OPEN unter-

lassen, wird vom Interpreter eine Datensatzlänge von 128 Byte vorgesehen. Weitere Informationen zu diesem Thema finden Sie unter 5.5.1 "Funktionsweise einer Random-Access-Datei".

In Version V3.0 kann die Eingabe der OPEN-Zeile extrem verkürzt werden. So wird z.B. o I 1 Name.Lst vom Editor in OPEN "I",#1, "Name.Lst" und z.B. o o 1 Name$ in OPEN "o",#1,Name$ umgewandelt. Alle An- und Ausführungsstriche, das Nummernzeichen, sowie die Trennkommas können vernachlässigt werden, sofern zwischen den einzelnen Komponenten ein Leerzeichen angegeben wird.

PUT # { PU } — Datensatz schreiben

PUT #Kanal [,Satznummer]

Es wird der durch Satznummer definierte Datensatz aus den mit FIELD spezifizierten String-Variablen in die R-Datei mit der Nummer #Kanal geschrieben. Wird keine Satznummer angegeben, wird der jeweils nächste - bzw. in V3.0 der durch RECORD ggfs. gesetzte - Datensatz gelesen. Genauere Informationen hierzu finden Sie unter 5.5.1 "Funktionsweise einer Random-Access-Datei".

Version 3.0

RECALL { RECA } — String-Feld aus Datei lesen

RECALL #Kanal,Feld$(),Anz,Zeilenvar

In Verbindung mit STORE# ein wahrlich gewaltiger Befehl, der es zum Vergnügen werden läßt, z.B. eine Textverarbeitung zu programmieren. Wer sich schon einmal mit einem solchen Programm angelegt hat, wird wissen, was ich meine.

RECALL liest Anz Textzeilen aus der zuvor geöffneten Datei mit der Kennung Kanal in das String-Feld Feld$(). Carriage Return (CHR$(13))/Line Feed (CHR$(10) wird dabei als Zeilenendemarkierung interpretiert.

Elementen des Feldes werden der Reihe nach ab Element 0 (OPTION BASE 0) bzw. ab Element 1 (OPTION BASE 1) je eine Zeile zugeordnet. Ist das Feld zu kurz (die Elementeanzahl ist dann kleiner als die Anzahl gelesener Zeilen), wird der Leseprozess beim letzten Element ohne Meldung abgebrochen. Wird das File-Ende erreicht, wird ebenfalls ohne Meldung abgebrochen. Zeilenvar ist eine numerische Rück-

gabevariable (Fließkomma- oder 32-Bit-Integer), die nach Abschluß die Anzahl der tatsächlich gelesenen Zeilen enthält.

Der Lesezeiger bleibt nach RECALL bei nicht geschlossenen Dateien auf dem Anfang der nächsten Zeile stehen. Das heißt, daß bei weiteren RECALL-Aufrufen ab dieser File-Position, bzw. nur noch der verbleibende Dateirest gelesen wird. Soll wieder ab Datei-Beginn gelesen werden, ist der File-Pointer durch SEEK #Kanal,0 wieder auf den Datei-Anfang zu richten.

Beispiel:

```
Open "O",#1,"AR_STORE.DAT"      ! Output-Datei öffnen
Dim Sfeld$(6)                   ! String-Feld einrichten
For I%=0 To 5                   ! 6 Strings
  Read Sfeld$(I%)               ! lesen
Next I%
Store #1,Sfeld$(),6             ! STORE String-Feld
Close #1                        ! Datei schließen
Erase Sfeld$()                  ! Feld löschen
Open "I",#1,"AR_STORE.DAT"      ! Input-Datei öffnen
Dim Rfeld$(4)                   ! Neues String-Feld anlegen
Recall #1,Rfeld$(),4,Back%      ! 4 Strings zurücklesen
Print "Anzahl gelesener Zeilen : ";Back%
For I%=0 To Back%               ! Gelesene Strings...
  Print Rfeld$(I%)              ! ausgeben
Next I%
Erase Rfeld$()                  ! Feld löschen
Dim Rfeld$(10)                  ! und neu einrichten
Recall #1,Rfeld$(),100,Back%    ! Dateirest lesen
Print "Anzahl gelesener Zeilen : ";Back%
For I%=0 To Back%               ! Restliche Zeilen...
  Print Rfeld$(I%)'             ! ...ausgeben
Next I%
Close #1                        ! Datei schließen
Data zeile1,zeile2,zeile3,zeile4,zeile5,zeile6
```

Version 3.0

RECORD { REC } Satz-Pointer für GET#/PUT# setzen

RECORD [#] Kanal,Satznummer

Setzt den Satz-Zeiger für den nächsten GET#- oder PUT#-Zugriff auf eine R-Datei. Wird beim nächsten GET# oder PUT# der Parameter "Satznummer" ausgelassen, wird der durch RECORD# aktualisierte Satz gelesen bzw. geschrieben.

RELSEEK { REL } File-Pointer verschieben

RELSEEK #Kanal,[-] Offset

RELSEEK (relativ Seek) verschiebt den File-Pointer der Datei mit der Kennung #Kanal relativ zur aktuellen Pointer-Position entweder in Richtung File-Ende oder File-Anfang um die mit "Offset" bestimmte Anzahl von Bytes. Im zweiten Fall (Richtung File-Anfang) ist dem Offset-Wert ein Minuszeichen voranzustellen.

Beispiel:

```
Open "U",#1,"Dateiname"  ! Datei öffnen
Seek #1,Int(Lof(#1)/2)   ! Der File-Pointer zeigt nun
'                        ! auf das Byte in der Dateimitte
Relseek #1,3             ! Der Zeiger wird in Richtung File-Ende
'                        ! um drei Byte weitergesetzt
Relseek #1,-6            ! Der Zeiger wird in Richtung File-Anfang
'                        ! sechs Byte zurückgesetzt. Er befindet
'                        ! sich nun 3 Byte vor der durch SEEK
'                        ! bestimmten Mittelposition.
'... etc.                ! Weiteres Programm
```

Bei den Befehlen SEEK und RELSEEK ist darauf zu achten, daß nicht versucht wird, den Zeiger auf eine Position zu setzen, die in der angegebenen Datei nicht vorhanden ist, bzw. nicht vorhanden sein kann (kleiner Null oder größer File-Ende). Wird dies versucht, erscheint die Fehlermeldung "Seek falsch?".

SEEK { SEE } File-Pointer setzen

SEEK #Kanal,[-] Offset

Mit diesem Befehl kann der File-Pointer der Datei mit der Kennung "#Kanal" auf ein durch "Offset" bestimmtes, beliebiges Byte gesetzt werden. Diese neue Position bezieht sich entweder absolut auf den File-Anfang oder auf das File-Ende. Im zweiten Fall (relativ zum File-Ende) ist dem 'Offset'-Wert ein Minuszeichen voranzustellen. Eine Anwendungsmöglichkeit finden Sie im RELSEEK-Beispiel bzw. im RAMKART-Listing/Procedure Datload.

Version 3.0

TOUCH { TOU } Datei-Zeiteintrag aktualisieren

TOUCH #Kanal

Schreibt die aktuelle System-Zeitangabe (siehe TIME$) in die dafür vorgesehene Position des betreffenden Directory-Eintrags. #Kanal gibt dabei die Kennung der durch OPEN geöffneten Datei an. Im ersten Beispiel zu FILES finden Sie eine Möglichkeit, allein die Dateizeit aus dem Direktory-Eintrag auszufiltern.

Version 3.0

STORE { STOR } String-Feld in Datei ablegen

STORE #Kanal,Feld$() [,Anz]

STORE speichert die Elemente eines durch Feld$() bestimmten String-Feldes der Reihe nach in der Datei mit der Kennung #Kanal. Als Endmarkierung wird jeder geschriebenen Zeile ein CR/LF (Carriage Return/Line Feed = CHR$(13)/CHR$(10)) angehängt. Durch den optionalen Parameter Anz kann bestimmt werden, wie viele Elemente maximal in der Datei gesichert werden sollen. Wird dieser Parameter ausgelassen, wird das komplette Feld gespeichert.

Außerdem kann STORE auch für virtuelle Dateien (AUX:, PRN: etc.) verwendet werden. Bei RECALL ist das jedoch nicht möglich.

Ein Beispiel hierzu finden Sie unter RECALL#.

5.5.1 Funktionsweise einer Random-Access-Datei

Wie die Behandlung von Variablenfeldern eine Sonderstellung einnimmt, so sind auch die Befehle zur Behandlung von Random-Access-Dateien (random access = engl.: wahlfreier Zugriff) eine besondere Erklärung wert.

Alle anderen Zugriffsvarianten auf Disketten- bzw. Hard-Disk-Dateien arbeiten grundsätzlich seriell, d.h.: der Reihe nach. Es wird also eindimensional Zeichen für Zeichen nacheinander gelesen, bzw. geschrieben. Diese Reihenfolge ist nur durch SEEK und RELSEEK veränderbar.

Eine R-Datei ist vergleichbar mit einem zweidimensionalen Speicherfeld. Es können mehrere Datenblöcke unter einem gemeinsamen Oberbegriff - hier der Dateiname - zusammengefaßt werden. Während die Elemente eines Variablenfeldes durch die Indizes repräsentiert werden, können hier die Datenblöcke (Elemente der 1. Dimension) durch die Satznummer angesprochen werden. Innerhalb eines Blocks befinden sich dann die zusammengehörigen Einträge (Elemente der 2. Dimension). Um eine solche Datei anzulegen, muß sie zuerst geöffnet werden. Ihr Sonderstatus wird dabei durch den Arbeitsmodus R kenntlich gemacht.

```
Open "R",#1,Dateiname,Satzlänge
```

Nachdem dieses getan wurde, kann mit dem Befehl FIELD bestimmt werden, in wieviel einzelne Einträge ein Satz unterteilt sein soll. Gleichzeitig wird angegeben, wieviele Zeichen (Bytes) jedem einzelnen dieser Einträge zuzuteilen sind. Da die Länge des gesamten Datensatzes durch die Angabe von Satzlänge im OPEN-Befehl bestimmt wird, ist es einleuchtend, daß die Summe der Zeichen der angegebenen Einträge insgesamt nicht größer sein darf als die Satzlänge. Sie darf aber kleiner sein. Die einmal so bestimmte Größe eines Satzes und seine Einteilung sollte von nun an nicht mehr verändert werden. Tun Sie es dennoch, ist die korrekte Übergabe der Daten an die zugeordneten Variablen nicht mehr gewährleistet. Natürlich können Sie neue Datenfelder in einen neu definierten Datensatz mit demselben Dateinamen einsetzen, nur dann sind die vorher darin enthaltenen Daten nicht mehr zu erreichen.

Zum anderen dürfen die durch PUT# in die Feldeinteilung geschriebenen Datensätze in ihrer Größe nicht mehr variiert werden, da sonst Daten des folgenden Satzes mit Daten des aktuellen Satzes kollidieren. Um die festgelegten Satz- und Eintragslängen immer einzuhalten, werden zur Vorbereitung der Einträge die Befehle LSET und RSET verwendet. Diese fügen die übergebenen Texte in die definierten FIELD-String-Variablen ein, ohne deren Länge zu verändern.

Es ist problemlos möglich, eine einmal so installierte Random-Access-Datei zu erweitern oder zu ändern, indem man der Datei einfach weitere Sätze per PUT#-Anweisung anhängt bzw. in diese einfügt. Einmal geschriebene Sätze lassen sich jedoch nicht bzw. nur über Umwege wieder entfernen. In Version V3.0 können Sie eine solches File komplett über RECALL einlesen, dann durch INSERT oder DELETE bearbeiten und abschließend durch STORE wieder zurückschreiben. In V2.xx bleibt Ihnen nur noch die Möglichkeit, über INPUT#, BLOAD,

BGET# oder ähnliche Befehle das File zu lesen und dann durch BMOVE, SWAP etc. zu modifizieren.

Erweiterungen innerhalb des Satzgefüges (z.B. zusätzliche Einträge) sind nur möglich, indem eine neue Datei mit anderem FIELD-Aufbau eingerichtet wird und die Datensätze der alten Datei nach dem Eimerketten-Prinzip gelesen, erweitert und dann komplett an die neu eingerichtete R-Datei weitergeleitet werden.

Achten Sie darauf, daß bei jedem OPEN für eine schon eingerichtete Datei dieser Art immer exakt dieselbe Datensatzlänge angegeben wird, bzw. die neu angegebene Satzlänge größer als die vorhandene ist. Wird nämlich keine Satzlänge genannt, wird diese vom Interpreter selbsttätig auf 128 Bytes festgelegt. Falls Sie dann die angesprochene Datei vorher größer dimensioniert hatten, wird beim Versuch, ein Feld anzusprechen, das mit seiner Länge über die angegebene Satzlänge hinausreicht, die Fehlermeldung "File-Ende erreicht" ausgegeben.

Eine Grafik soll das Prinzip verdeutlichen:

```
 ________________________________________________________________________
|<       1. Datensatz        >|<       2. Datensatz        >|< n. Datensatz
|<'Satzlänge' z.B. 45 Byte>   |<     Satzlänge wie vor     >|
|                             |                             |
|_____________________________|_____________________________|_____________
|Eintrag1 |Eintrag2 |Eintrag3 |Eintrag1 |Eintrag2 |Eintrag3 |
|  z.B.   |  z.B.   |  z.B.   |wie vor: |wie vor: |wie vor: |   ... ---.
|10   Byte|20   Byte|15   Byte|10   Byte|20   Byte|15   Byte|   ...    |->
|         |         |         |         |         |         |   ... ---'
|_________|_________|_________|_________|_________|_________|_____________
```

Die Anzahl der Sätze, die Größe der einzelnen Einträge, sowie die Satzlänge können von Ihnen frei bestimmt werden. Eingeschränkt werden Sie nur durch eine maximale Datensatzlänge von 65535 Byte, durch den Umstand, daß die Summe aller Eintragslängen die angegebene Satzlänge nicht überschreiten darf und - in V2.xx - durch die maximale Programmzeilenlänge für eine FIELD-Anweisung (max. 255 Zeichen pro FIELD). Numerische Werte sind in V2.xx vor ihrer Speicherung in einer R-Datei durch MKI$/MKL$/MKS$/MKF$/MKD$ in eine Zeichenkette umzuwandeln. In V3.0 können numerische Werte auch direkt übergeben werden (siehe FIELD AT).

Zum tieferen Verständnis folgt nun ein Beispielprogramm, daß Ihnen als Anregung zu eigenen Experimenten dienen soll.

Dieses Programm besteht aus vier voneinander unabhängigen Blöcken. Im ersten Block wird eine Random-Access-Datei mit einer Satzlänge

von 90 Byte installiert. Diese 90 Byte werden mit der FIELD-Anweisung in drei Einträge (40/30/20 Byte) unterteilt. Gleichzeitig werden die für die Aufnahme der Einträge vorgesehenen String-Variablen mit Leerzeichen gefüllt. Im Anschluß daran werden in der FOR/NEXT-Schleife sechs Datensätze eingelesen.Nach der Eingabe der jeweiligen Einträge werden die Inhalte der INPUT-Variablen durch LSET linksbündig in die vorgesehenen - und vorbereiteten - Puffer-Strings eingesetzt. Zu guter Letzt wird dann dieser Datensatz mit PUT# unter Angabe der Datensatznummer (hier der Schleifenindex I%) in die Datei geschrieben.

Block 2 demonstriert das Erweitern und Ändern einer bestehenden Datei. Hier sind die Texte willkürlich vorgegeben und können natürlich von Ihnen frei bestimmt werden Die letzen beiden Programmblöcke zeigen zwei Möglichkeiten, die einzelnen Sätze aus der Datei wieder herauszulesen und auszugeben.

Beispiel:

```
' Schreib-Variante 1:
' (Datei-Erröffnung)
'---------------------
Open "R",#1,"Adress.Dat",90 ! R-Datei erstmals öffnen
Field #1,40 As Zeile.1$,30 As Zeile.2$,20 As Zeile.3$
'                           ! Satz einrichten
For I%=1 To 6               ! 6 Sätze >---------------------.
  Print At(20,11);"Name    :";                              !
  Form Input 40,N$          ! Eintrag 1 eingeben            |
  Print At(20,12);"Vorname :";                              !
  Form Input 30,V$          ! Eintrag 2 eingeben            |
  Print At(20,13);"Telefon :";                              !
  Form Input 20,T$          ! Eintrag 3 eingeben            |
  Cls                       ! Bildschirm klar               |
  Lset Zeile.1$=N$          ! Eintrag 1 linksbündig einfügen |
  Lset Zeile.2$=V$          !    -"-   2      -"-      -"-  |
  Lset Zeile.3$=T$          !    -"-   3      -"-      -"-  |
  Put #1,I%                 ! Datensatz in R-File eintragen |
Next I%                     ! nächster Satz <---------------'
Close #1                    ! R-File schließen
'     Schreib-Variante 2 :
' (nachträgliches An-/Einfügen)
'----------------------------
Open "R",#1,"Adress.Dat",90 ! Öffnen (Satzlänge beachten)
Field #1,40 As Zeile.1$,30 As Zeile.2$,20 As Zeile.3$
  '                           ! Satzeinteilung wie oben
Lset Zeile.1$="Satz 7/Zeile 1" ! Eintrag 1 linksbündig
Lset Zeile.2$="Satz 7/Zeile 2" !   -"-   2     -"-
Lset Zeile.3$="Satz 7/Zeile 3" !   -"-   3     -"-
   Put #1,7                    ! neuen Datensatz nachtragen
Lset Zeile.1$="Satz 2/Zeile 1" ! Eintrag 1 linksbündig
Lset Zeile.2$="Satz 2/Zeile 2" !   -"-   2     -"-
Lset Zeile.3$="Satz 2/Zeile 3" !   -"-   3     -"-
   Put #1,2                    ! 2. Datensatz überschreiben
```

```
  Close #1                       ! und wieder schließen
'  Lese-Variante 1:
' (seriell Einlesen)
'-------------------------
Open "R",#1,"Adress.Dat",90 ! Öffnen (Satzlänge beachten)
Field #1,40 As Zeile.1$,30 As Zeile.2$,20 As Zeile.3$
'                           ! Satzeinteilung wie oben
For I%=1 To 7               ! 7 Datensätze >------------.
  Get #1,I%                 ! lesen und die Einträge den |
  '                         ! FIELD-Variablen zuweisen   |
  Print "Name     : ";Zeile.1$ ! Eintrag 1 ausgeben      |
  Print "Vorname  : ";Zeile.2$ !   -"-   2   -"-         |
  Print "Telefon  : ";Zeile.3$ !   -"-   3   -"-         |
  Print                     ! Leerzeile                  |
Next I%                     ! Nächster Satz <-----------'
Close #1                    ! File schließen
Cls                         ! Bildschirm klar
Pause 200                   ! Kleine Pause
'  Lese-Variante 2:
' (wahlfrei Einlesen)
'-------------------------
Open "R",#1,"Adress.Dat",90 ! Öffnen (Satzlänge beachten)
Field #1,40 As Eintrag1$,30 As Eintrag2$,20 As Eintrag3$
'                           ! Satzeinteilung wie gehabt
Do                          ! Lese-Schleife >----------------.
  Print At(20,11);"Satznummer eingeben (0=Abbruch): ";       |
  Repeat                    ! Eingabe-Schleife >--------.    |
    Satz%=Val(Chr$(Inp(2))) ! Satznummer holen          |    |
  Until Satz%<8             ! Bis gültige Satznummer <--'    |
  Exit If Satz%=0           ! A-Z-Taste oder 0 gedrückt?     |
  Get #1,Satz%              ! Gewünschten Datensatz lesen    |
  Print At(20,10);"Name     : ";Eintrag1$ ! Eintrag 1 ausgeben|
  Print At(20,11);"Vorname  : ";Eintrag2$ !   -"-   2   -"-   |
  Print At(20,12);"Telefon  : ";Eintrag3$ !   -"-   3   -"-   |
  Print At(20,15);"<Taste>"                                  |
  Repeat                    ! Warte ... >-------.            |
  Until Len(Inkey$)         ! ... auf Taste <---'             |
  Cls                       ! Bildschirm klar                |
Loop                        ! Wieder von vorn  <-------------'
```

5.6 Port-Ein-/-Ausgabebefehle

INP? — Port auf Empfangsbereitschaft testen

Var=INP?(Port)

Teilt mit, ob am angegebenen Peripherie-Port ein Byte anliegt. Port bestimmt die Kennziffer der zu prüfenden Schnittstelle (siehe INP). Kann ein Byte gelesen werden, wird der Wert -1 (True) zurückgegeben. Andernfalls der Wert 0 (False).

INP — Daten byteweise von Peripherie lesen

Var=INP(Port)
Var=INP(#Kanal)

Dieser Funktion sind Sie in den bisherigen Beispielprogrammen schon begegnet. Sie eignet sich in diesen Fällen dazu, den Programmlauf zu unterbrechen und auf einen Tastendruck zu warten (VOID INP(2)). Ihre eigentliche Funktion ist es jedoch, jeweils ein einzelnes Byte von einem Peripherie-Port oder aus einer Datei zu lesen.

In der nachgestellten Klammer wird entweder die Nummer des Datenkanals oder eine Identifikationsziffer für den jeweils gewählten Port übergeben. Dabei ist Vorsicht geboten. Dieser Befehl läßt sich in seinem Eifer, auf ein Byte zu warten auch nicht von der Break-Funktion <Control><Shift><Alternate> beirren. Wurde er aufgerufen, ohne daß auf dem angegebenen Port ein Byte anliegt, hilft nur noch der Griff zum Reset-Knopf.

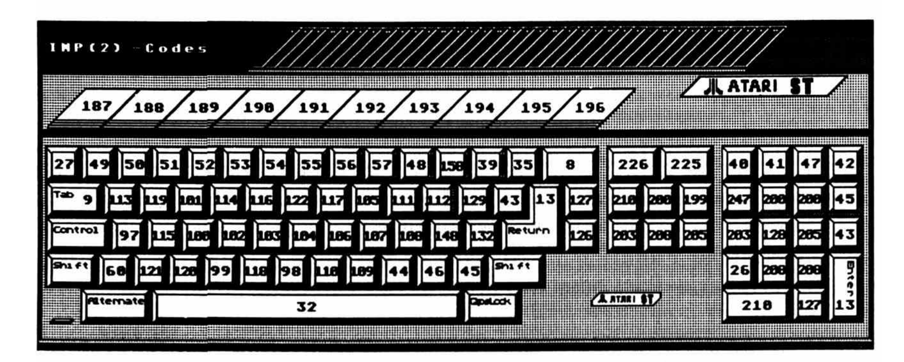
Die Ports haben folgende Identifikatoren:

```
0   LST:   (Drucker-Port bzw. Centronics-Port)
1   AUX:   (Auxilliary-Port bzw. RS232)
            In V3.0 siehe auch INPAUX$
2   CON:   (Consol-Port = Tastatur)
3   MID:   (MIDI-Port)
            In V3.0 siehe auch INPMID$
4   IKB:   (Keyboard-Prozessor)
5   VID:   (Video-Port = Monitor)
```

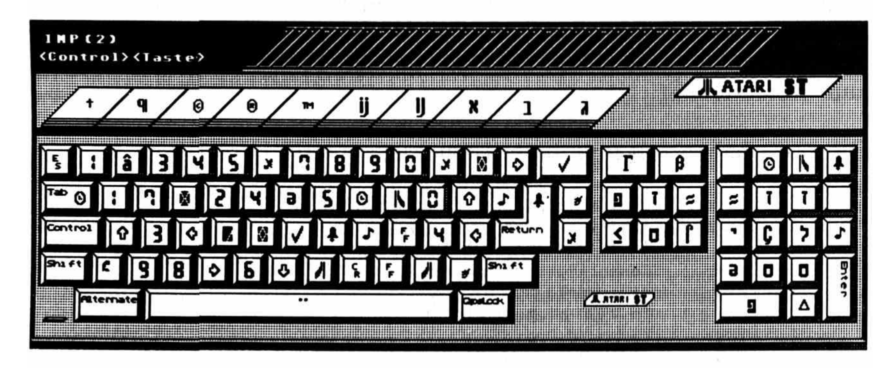
Beispiele:

```
X=Inp(2)
```

Wartet auf einen Tastendruck und speichert den ASCII-Wert der gedrückten <Taste> in X. GFA-INP(2) ist kompatibel zu ST_BASIC-INP(2).



Wird zur gedrückten <Taste> zusätzlich <Control> gedrückt, ergibt sich eine andere Belegung.

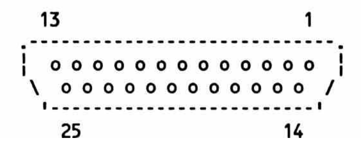


X=Inp(3) Wartet darauf, daß ein Byte am MIDI-Port anliegt und speichert dessen Wert in X (MIDI = Musical Instruments Digital Interface).

Out 5,Inp(#1):
Gibt den ASCII-Wert des an File-Pointer-Position stehenden Zeichens der Datei mit der Kanalnummer #1 auf dem Bildschirm aus.

Var=Inp(0):
Liest ein Byte vom Centronics-Port. Normalerweise ist dieser Port als Ausgabe-Schnittstelle für einen Drucker vorgesehen. Durch einen kleinen Trick läßt sich der ST mit einem anderen Computer verbinden, solange dieser ebenfalls über einen Centronics-Port verfügt.

Centronics-Pinbelegung:



```
1    = STROBE zeigt an, daß die anliegenden
       Daten gültig sind.
2  --.
bis  |- DATA. 8-Bit-Datenleitung
9  --'
10   = Nicht belegt
11   = BUSY-Signal. Wird vom Drucker gesendet,
       wenn sein Puffer voll oder das Gerät
       Offline ist.
12 --.
bis  |- Nicht belegt
17 --'
18 --.
bis  |- GROUND.
25 --'
```

Soll der ST vom zweiten Computer aus als Drucker angesprochen werden, sind Pin 1 und Pin 11 der beiden Stecker über Kreuz zu legen. Alle anderen Pins werden direkt angeschlossen. Werden vom anderen Computer Druckerausgaben gesendet, können sie nun durch INP(0) vom ST gelesen werden.

Var=Inp(3)

Liest ein Byte vom MIDI-IN-Port. Die beiden MIDI-Ports (In/Out) eignen sich zum Austausch von Daten zwischen ST und einem zweiten MIDI-Gerät. Üblicherweise ist dieses zweite Gerät ein Synthesizer, der die gesendeten Daten in Musik übersetzt.

Dazu sind allerdings ausgiebige Kenntnisse des MIDI-Datenstandards von Nöten. Im anderen Fall kann das zweite Gerät auch ein zweiter ST sein, wobei der MIDI-Out-Port des einen mit dem MIDI-In-Port des anderen ST's durch ein handelsübliches 5-Pol-Diodenkabel verbunden wird. In V3.0 haben Sie durch INPMID$ eine komfortable Möglichkeit, den MIDI-Puffer auszulesen. In V2.xx sind Sie darauf angewiesen, den Puffer wie folgt zu lesen:

```
Clr Midin$
While Inp?(3)
  Midin$=Midin$+Chr$(Inp(3))
Wend
```

Zum Schreiben von größeren Datenmengen in den MIDI-Out-Port steht Ihnen XBIOS-Funktion 12 (midiws) zur Verfügung:

```
Midout$="beliebiger MIDI-String"
Void Xbios(12,Len(Midout$)-1,Varptr(Midout$))
```

Version 3.0

INPAUX$ — Text-String vom seriellen Port (RS232) lesen

Var$=INPAUX$

Reservierte String-Variable, die nach Aufruf alle - seit dem letzten Lesen - am seriellen Port (RS232) eingegangenen Zeichen enthält. Der Auxilliary-Puffer ist nach einem INPAUX$-Aufruf leer. Zuweisungen an diese Variable sind nicht möglich, bzw. bei der mir vorliegenden V3.0-Version sogar absturzgefährlich. In den Versionen V2.xx kann

derselbe Effekt - nur wesentlich langsamer - durch INP() erreicht werden.

Beispiel:

```
   AUX lesen:                 |  AUX schreiben:
                              |
Clr Aux$                      | For I%=1 To Len(Aux$)
While Inp?(1)                 |   Exit If Out?(1)=False
  Aux$=Aux$+Chr$(Inp(1))      |   Out 1,Asc(Mid$(Aux$,I%,1))
Wend                          | Next I%
Print Aux$                    | Aux$=Right$(Aux$,Len(Aux$)+1) ! evtl.
                              |       nicht ausgegebener String-Rest
```

Version 3.0

INPMID$ — Text-String vom MID-IN-Port lesen

Var$=INPMID$

Reservierte String-Variable, die nach Aufruf alle - seit dem letzten Lesen - am MIDI-IN-Port eingegangenen Zeichen enthält. Der MIDI-Puffer ist nach einem INPMID$-Aufruf leer. Zuweisungen sind nicht möglich (siehe INPAUX$).

Die Lese- und Schreibmöglichkeit über den Midi-Port für die V2.xx-Versionen finden Sie unter INP().

OUT? — Port auf Ausgabebereitschaft testen

Var=OUT?(Port)

OUT? liefert Information darüber, ob der angegebene Peripherie-Port ausgabebereit ist. Port bestimmt die Kennziffer der zu prüfenden Schnittstelle (siehe INP). Kann ein Byte ausgegeben werden, wird der Wert -1 (True) zurückgegeben. Andernfalls der Wert 0 (False).

OUT { OU } Daten byteweise an Peripherie ausgeben

OUT #Kanal,Bytewert
OUT Port,Bytewert
OUT #Kanal,Byte1 [,Byte2 [,Byte3,...]] **(nur V3.0)**
OUT Port,Byte1 [,Byte2 [,Byte3,...]] **(nur V3.0)**

Der Befehl OUT stellt das Gegenstück zu INP() dar. Hiermit lassen sich einzelne Byte-Werte an einen bestimmten Datenkanal oder Port senden.

Beispiele:

Out 5,122
Gibt das Zeichen z (ASCII 122) an der aktuellen Cursor-Position auf dem Monitor aus. Somit lassen sich auch Steuerzeichen (ASCII 0 - 31) beliebig auf dem Bildschirm plazieren.

Out #1,65
Schreibt den ASCII-Wert des Zeichens A (ASCII 65) an die aktuelle File-Pointer-Position der Datei mit der Kanalnummer 1.

Out 4,x IKB:-Steuerung:
Die an den Keyboard-Prozessor gerichteten Befehle ermöglichen verschiedene Einstellungen zur Maus-, Joystick- und Tastaturkontrolle.

Da dies ein sehr kompliziertes und umfangreiches Thema ist und es außerdem an ausreichender Dokumentation mangelt, kann ich Ihnen hier nur eine kleine Auswahl dieser Befehle bieten.

Vorsicht: IKB bedankt sich bei falscher Bedienung mit unvorhersehbaren - und oft auch irreparablen - Fehlfunktionen.

Bevor Sie jedoch zum Reset-Knopf greifen, versuchen Sie durch OUT 4,8 (evtl. auch mehrfach) den IKB: (Intelligent Keyboard) wieder einzufangen. Sollte das erfolglos bleiben, ist ein Reset allerdings meist unumgänglich.

Out 4,7

Maustasten-Reaktion einstellen. Es kann bestimmt werden, ob der Maustastenstatus ständig (Standard) oder nur bei Mausbewegungen geliefert wird. Im zweiten Fall werden die Maustasten außerdem als ASCII-liefernde Tasten gewertet.

Die linke Taste liefert bei INP(2) den Wert 244 und die rechte den Wert 245. Eine INP()-Abfrage kann also auch durch die Maustasten bedient werden. Es wird anschließend ein Parameter-Byte erwartet.

```
Out 4,1  ->  Ständige Meldung
Out 4,6  ->  Meldung nur bei Bewegung
Werte größer 6 sind unzulässig.
```

Out 4,8

Mauskontakt einschalten. Die Maus ist beweglich und meldet in MOUSEX/MOUSEY die Mauskoordinaten (hoffentlich!). Eigentlich hat dieser Befehl die Aufgabe, den relativen mousevec-Modus (siehe XBIOS(34)) zu aktivieren und somit die relativen (auf die vorherige Position bezogenen) X- und Y-Pixelabstände der neuen Mausposition zu liefern.

Da dieser Modus jedoch bei System-Start automatisch aktiv ist, kann man OUT 4,8 auch als Maus-Retter einsetzen. Mit dem folgenden kleinen Programm kann die Ablage-Adresse des mousevec ermittelt werden und aus dieser Adresse (+1 bzw. +2) dann die Offsets gelesen werden. Ist die neue Mausposition kleiner als die alte, ergibt sich der Offset aus 255 - Offset.

```
CLR Maus%                    ! Adresse klar
Move_a0_Adress$=MKI$(&H23C8)+MKL$(*Maus%)!-.
Rts$=MKI$(&H4E75)                           !- Maschinen-Code
Rout$=Move_a0_Adress$+Rts$ !---------------'
'
Vector%=XBIOS(34)+16         ! mousevec-Adresse holen
Old%=LPEEK(Vector%)          ! alten Zeiger sichern
LPOKE Vector%,VARPTR(Rout$)! eigene Routine einbinden
WHILE Maus%=0                ! auf Interrupt warten
WEND                         ! (erst nach Mausbewegung)
LPOKE Vector%,Old%           ! alten Zeiger restaurieren
' Der bisherige Teil ist vorzugsweise bei Programmstart
' auszuführen. Die folgende Abfrage kann dagegen an
' beliebigen Programmstellen eingesetzt werden.
REPEAT
  IF PEEK(Maus%+1) AND PEEK(Maus%+2)
    PRINT "X-/Y-Offset : ";PEEK(Maus%+1),PEEK(Maus%+2)
  ENDIF
UNTIL MOUSEK
```

Out 4,9

Absolute Mauskoordinaten liefern. Es werden vier weitere Parameter-Bytes erwartet.

```
Out 4,Hi-Byte der max. X-Meldung
Out 4,Lo-Byte der max. X-Meldung
Out 4,Hi-Byte der max. Y-Meldung
Out 4,Lo-Byte der max. Y-Meldung
```

Größere Werte, als durch die X- und Y-Maxima vorgegeben wurden, werden ignoriert. Ich habe versucht, die X- und Y-Koordinaten durch das obige mousevec-Verfahren zu ermitteln. Dies ist mir allerdings nicht gelungen. Die Maus blieb auch nach OUT 4,8 weiter inaktiv. Also Vorsicht!

Out 4,10

Die Bewegungen der Maus werden als Cursor-Tasten-Druck interpretiert. Zwei Parameter sind nachzusenden, die die zurückgelegte Pixel-Anzahl je Cursor-Schritt bestimmen.

```
Out 4,'X-Relation'
Out 4,'Y-Relation'
In V3.0: OUT 4,11,X,Y
```

Druck auf die linke Maustaste hat dann im GFA-Editor die Wirkung wie <Control><Cursor-Pfeil-rechts>. Dieser Modus wird durch OUT 4,8 wieder verlassen.

Out 4,11

Raster für Mausmeldungen einstellen. Es sind zwei Parameter nachzusenden, die das Raster beschreiben, innerhalb dessen die Mauskoordinaten gemeldet werden.

```
Out 4,'X-Raster'
Out 4,'Y-Raster'
In V3.0: OUT 4,11,X,Y
```

Rasterwert 1 stellt den Standard-Zustand wieder her.

Out 4,12

Maus-Skalierung. Anwendung ist unklar, da dieser Befehl nur im Absolutmodus verarbeitet wird (siehe OUT 4,9).

Out 4,13

Absolute Mausposition melden (keine Parameter). Anwendung ist unklar, da auch hier der mousevec-Modus nur die X/Y-Offsets liefert (siehe OUT 4,8).

Out 4,14

Koordinatenzähler setzen. Es werden 5 Parameter erwartet:

```
Out 4,0         ! 1 Null-Byte
Out 4,Hi-Byte der neuen X-Koordinate
Out 4,Lo-Byte der neuen X-Koordinate
Out 4,Hi-Byte der neuen Y-Koordinate
Out 4,Lo-Byte der neuen Y-Koordinate
```

Auswirkungen dieses Befehls konnte ich nicht feststellen. Dieser Modus wird durch OUT 4,8 wieder verlassen.

Out 4,15

Pixel-Koordinate 0/0 liegt links unten.

Out 4,16

Pixel-Koordinate 0/0 liegt links oben (Standard).

Out 4,17

Tastatur-Kontrolle nach OUT 4,19 wieder einschalten.

Out 4,18

Mauskontakt ausschalten. Die Maus ist unbeweglich und es werden keine Koordinaten mehr gemeldet. Dieser Modus wird durch OUT 4,8 wieder verlassen.

Out 4,19

Tastatur-Kontrolle ausschalten. INP(), INKEY$ etc. bleiben anschließend ohne Ergebnis, da der Kontakt von Tastatur zum Puffer komplett gesperrt ist.

Out 4,20

Maus-Port 0 als Joystick-Port interpretieren. In Adresse 3592 (bei ROM-TOS) wird ab jetzt ständig der Joystick-Status von Port 0 abgelegt (in V3.0 siehe STICK(0)). Im MEGA-TOS oder RAM-TOS müssen die Ablage-Adressen nach demselben Schema ermittelt werden, wie bei OUT 4,8 beschrieben wurde. Ändern Sie dazu bitte den mittleren Block folgendermaßen:

```
OUT 4,22
vector%=XBIOS(34)+24          ! Mousevec-Adresse holen
old%=LPEEK(vector%)           ! Alten Zeiger sichern
LPOKE vector%,VARPTR(rout$)! Eigene Routine einbinden
WHILE maus%=0                 ! Auf Interrupt warten
WEND                          ! (erst nach Mausbewegung)
LPOKE vector%,old%            ! Alten Zeiger restaurieren
OUT 4,20
```

In Maus%+1 ist dann der Joystick-Status von Port 0 und in Maus%+2 der Status von Port 1 zu finden.

```
  5  1  9
   \ ¦ /
 4---0---8
   / ¦ \
  6  2  10
```

Wird zusätzlich der Fire-Button gedrückt, erhöht sich der Statuswert um 128 (in V3.0 siehe STRIG(0)). In Adresse 3593 (ROM-TOS) wird generell (auch ohne OUT 4,20) der Status von Port 1 abgelegt (in V3.0 siehe STICK(1)). Der Firebutton-Status hierzu ist allerdings getrennt davon in Adresse 3582 zu finden (in V3.0 siehe STRIG(1)). Werte ungleich 248 bedeuten, daß der Button gedrückt wird. Der Mausmodus wird durch OUT 4,8 wieder eingeschaltet.

Out 4,21

Joystick-Modus (OUT 4,20), sowie Maus ausschalten. Durch OUT 4,8 wird der Standard-Mausmodus restauriert.

Out 4,22

Einmalige Abfrage des Ports 0 im Joystick-Modus. Die Maus bleibt dabei aktiv. Adressen wie bei OUT 4,20.

Out 4,23

Joystick-Dauermeldung einschalten. Es wird ein Parameter-Byte erwartet:

```
Out 4, Melde-Verzögerung (in 1/10 Sekunden)
```

Anwendung wie bei OUT 4,20. In Maus%+1 wird der Firebutton-Status beider Ports geliefert (Bit 0 für Port 1 und Bit 1 für Port 0). In Maus%+2 liegt die Jostickstellung (Bits 0 - 3 für Port 1 und Bits 4 - 7 für Port 0).

Out 4,24

Firebutton-Dauermeldung einschalten (keine Parameter).

Out 4,25

Maus-Port 0 wird im Joystick-Modus interpretiert und die eingehenden Joystick-Bewegungen als Cursor-Tasten-Druck gewertet. Es sind sechs weitere Parameter nachzusenden.

```
Out 4,X-Startverzögerung (nur wenn Parameter 3 = 0)
Out 4,Y-Startverzögerung (nur wenn Parameter 4 = 0)
Out 4,X-Reaktionsverzögerung
Out 4,Y-Reaktionsverzögerung
Out 4,X-Wiederhol-Frequenz
Out 4,Y-Wiederhol-Frequenz
```

Durch OUT 4,8 wird der Standard-Mausmodus restauriert.

Out 4,26

Joystick-Dauermeldung ausschalten.

Out 4,27

Uhrzeit setzen (siehe SETTIME).

Out 4,28

Uhrzeit lesen (siehe TIME$).

Out 4,29

Tastaturspeicher bestimmen und belegen. In GFA-BASIC unbrauchbar.

Out 4,30
Tastaturspeicher lesen. In GFA-BASIC unbrauchbar.

Out 4,31
Programm im Tastaturspeicher starten. In GFA-BASIC unbrauchbar.

5.7 Drucker-Anweisungen

HARDCOPY {H} **Bildschirm auf Drucker ausgeben**

HARDCOPY

Der aktuelle Bildschirminhalt wird als Screen-Dump auf den Drucker übertragen. Dazu wird dieselbe System-Funktion verwendet, wie sie auch durch <Alternate><Help> auslösbar ist. Ist der Drucker nicht OnLine, nicht angeschaltet oder überhaupt nicht angeschlossen, wird das Programm nach ca. 30 Sekunden (TimeOut) fortgesetzt. Die Befehlsausführung läßt sich ausschließlich durch <Alternate><Help> unterbrechen. Die Break-Funktion <Control><Shift><Alternate> ist außer Kraft.

In diesem Zusammenhang ist ein kleiner Trick interessant. Das System löst generell eine Hardcopy immer dann aus, wenn der Word-Inhalt der Adresse &H4EE gleich Null ist. Anschließend wird ihr Inhalt um 1 vermindert. Wird nun <Alternate><Help> gedrückt, wird der Adressen-Inhalt durch das System um 1 erhöht, so daß sich der Wert Null ergibt und eine Hardcopy ausgeführt wird. Eine Hardcopy kann also auch durch SDPOKE &H4EE,0 ausgelöst werden (für nicht Epson-kompatible Drucker ggfs. die einzige Möglichkeit, eine Hardcopy zu produzieren).

Wird nun in &H4EE ein Wert größer Null ge'dpoked', wird der Zähler beim nächsten <Alternate><Help> wieder um 1 erhöht. Da sich daraus dann jedoch nicht der Wert 0 ergibt, wird keine Hardcopy ausgeführt. So haben Sie also die Möglichkeit, die Ausführung der Hardcopy-Routine zu unterdrücken. Ob in der Zwischenzeit vom Anwender versucht wurde, eine Hardcopy durch <Alternate><Help> auszulösen, erkennen Sie daran, daß der Inhalt von &H4EE größer ist als der Wert, den Sie vorher hineinge'dpoked' haben. In diesem Fall können Sie durch SDPOKE &H4EE,0 eine Hardcopy dann auslösen, wenn es Ihnen bzw. Ihrem Programm paßt. Anschließend an die Hardcopy wird der Zähler durch das System automatisch wieder auf 65535

(vorzeichenlos) bzw. auf -1 (vorzeichenbehaftet) gesetzt und das Spiel kann von vorn beginnen.

In der V3.0-Version vom 15.6.88 wird durch HARDCOPY generell ein SDPOKE &H4EE,0 und VSYNC ausgeführt, woraus sich ergibt, daß

a. auch nicht Epson-kompatible Drucker über HARDCOPY zum Ausdruck eines Screen-Dumps überredet werden können

b. das oben beschriebene SDPOKE &H4EE-Spielchen nicht mehr funktioniert, sobald der Befehl HARDCOPY inzwischen verwendet wurde. In diesem Fall wird ja generell die Adresse &H4EE auf Null gesetzt, womit nach Abschluß der Hardcopy wieder der Standardzustand eingestellt ist.

LLIST {LL} — Programm-Listing ausdrucken

LLIST

Soll das aktuelle Programm-Listing auf den Drucker übertragen werden, kann das LLIST erreicht werden (vgl. Editor-Funktion Llist). Die Ausführung dieses Befehls kann in V2.0 durch nichts gestoppt werden (auch nicht durch die GFA-Break-Funktion). Es sei denn, Sie schalten den Drucker OffLine oder ganz einfach aus. Nach ca. 30 Sekunden meldet sich dann Ihr Programm wieder.

Je nach Druckertyp- und marke kann diese Unterbrechung während des Druckvorgangs unterschiedliche Auswirkungen haben. Die meisten Drucker wird es nicht stören, wenn sie in ihrer Arbeit brutal unterbrochen werden. Einige jedoch könnten es Ihnen übel nehmen. Sie werden sicher schon wissen, ob ihr Drucker zu dieser mißmutigen Gattung gehört. Grundsätzlich ist jedoch bei einem Ausschalten des Druckers eine kurze Frist von ca. 3-5 Sekunden einzuhalten, bevor Sie das Gerät wieder einschalten. Diese Besinnungspause gehört sozusagen zur Wartung des Druckers. Er wird es mit einem langen Druckerleben vergelten.

In V2.02 und V3.0 kann das Drucker-Listing jederzeit (falls Break-Funktion aktiv) durch <Control/Shift/Alternate> unterbrochen werden. Der Interpreter ist dann wieder arbeitsbereit und es wird nur noch der Inhalt des Drucker-Puffers ausgedruckt. Außerdem werden in V3.0 die im Programm befindlichen Punkt-Befehle (siehe Erklärung zur Editor-Funktion Llist) auch hier berücksichtigt.

LPOS() Druckkopfposition ermitteln

Var=LPOS(Dummy)

Mit dieser Funktion können Sie den aktuellen Standort des virtuellen Druckkopfes im Zeichenpuffer Ihres Druckers ermitteln (max. 255). Da dieser nicht mit der aktuellen Position des physikalischen Schreibkopfes (der, der den Lärm macht) übereinstimmen muß, kann es unter Umständen wichtig sein, die gerade im Speicher aktuelle Position zu erfahren. Nach einem Carriage Return (LPRINT) beginnt die Zählung wieder bei Null. Es wird in Klammern ein beliebiger Wert übergeben, der für die Funktion ohne Bedeutung ist (dummy; engl. für: Attrappe/Schwindel).

Beispiel:

```
Lprint "VIRTUELLE DRUCKKOPF-POSITION"
For I=1 To 150
  Lprint Lpos(0)'
Next I
A=Lpos(0)
Print A
```

LPRINT {LPR} Daten an Drucker ausgeben

LPRINT [,'] "Text" [[;,'] Var1 [;,'] Expr...]

Grundsätzlich arbeitet LPRINT genauso wie PRINT. Die Syntax ist bis auf die AT(Xpos,Ypos)-Variante mit PRINT vergleichbar. Ein Unterschied ist, daß LPRINT die gewünschten Zeichen, Strings und Werte nicht auf dem Bildschirm, sondern auf dem Drucker ausgibt. Bezüglich der Drucker- und Druckkopf-Steuerung sollten Sie sich mit Ihrem Drucker-Handbuch eingehend auseinandersetzen. Dort finden Sie dann auch die notwendigen Steuerbefehle zur PRINT AT-Simulation etc.

Beispiel (für Epson-kompatible):

```
Lprint "Aktuelle Schrift"
Lprint Chr$(15);"Schmalschrift an  = CHR$(";15;")"
Lprint Chr$(18);"Schmalschrift aus = CHR$(";18;")"
Lprint Chr$(14);"Breitschrift  an  = CHR$(";14;")"
Lprint "Normalschrift"            ! Standard-Druck
Lprint Chr$(10);                  ! Puffer ausdrucken
```

Ein Grundproblem beim Ausdruck von Text ist immer, daß die einzelnen Drucker verschiedenen Standard-Zeichensätze zur Verfügung stellen. Man versteht zwar unter ASCII eine allgemein standardisierte

Codierung der einzelnen Buchstaben, aber wenn Sie schon versucht haben sollten, mit Ihrem Drucker Text auszudrucken, innerhalb dessen sich bestimmte Zeichen befinden (z.B. ß, ä, Ä, |, # etc.), so wird Ihnen aufgefallen sein, daß dort nicht unbedingt das gedruckt wird, was Sie auf dem Bildschirm sehen.

Ich möchte Ihnen deshalb hier zwei kleine Prozeduren anbieten, die es Ihnen ermöglichen, die undefinierten bzw. falsch definierten Zeichen an den ST-Zeichensatz anzpassen. Voraussetzung ist dazu, daß Sie einen Drucker besitzen, der mit Epson FX-80/FX-85/FX-105 Epson-kompatibel ist (z.B. Star NL-10). Ob die Epson-RX-Serie ebenfalls mit diesen Routinen bedient werden kann, konnte ich nicht überprüfen, da in meinem "Einzugsgebiet" nur mit Epson-FX-xx, P6 und Star NL-10 gearbeitet wird. Wenn Sie nicht genau wissen, ob Ihr Drucker zu den Epson-kompatiblen gehört, probieren Sie die Routinen einfach aus - es wird sich zeigen.

Die erste Routine stellt nur die Möglichkeit zur Verfügung, einzelne Zeichen grafisch zu gestalten. Sie legt dann das in dem Bitraster-String-Feld angegebene Bit-Muster im 11x8-Punktformat als DATA-Zeile in der Datei EPS_DATA.LST ab, die dann in die zweite Prozedur EPS_INIT eingemergt werden kann. Dort können die selbstdefinierten Zeichen gesammelt an den Drucker weitergeleitet werden.

Die Datei EPS_DATA.LST wird übrigens jedes Mal überschrieben. Sobald Sie also ein Zeichen in dieser Datei abgelegt haben, holen Sie es in den Arbeitsspeicher (Merge), falls Sie anschließend noch weitere Zeichen definieren möchten.

```
Dim D.str$(7)                ! Bit-Muster-Feld einrichten
Ascii%=35                    ! ASCII-Wert des zu ändernden
'                            ! Text-Zeichens
' -------------------------.
D.str$(0)="00000000000"      !
D.str$(1)="00110001100"      !    Vor Start
D.str$(2)="00110001100"      !    Bitmuster
D.str$(3)="01111111110"      !--- editieren.
D.str$(4)="00110001100"      !    8 Zeilen *
D.str$(5)="01111111110"      !   11 Spalten
D.str$(6)="00110001100"      !
D.str$(7)="00110001100"      !
' -------------------------'
Epdat$="D "+Str$(Ascii%)+","  ! DATA-Zeile vorbereiten
For Epi%=1 To 11             ! 11 Spalten
  Clr Sp%                    ! Byte-Puffer löschen
  For Epj%=0 To 7            ! 8 Nadeln (keine Unterlängen)
    Add Sp%,(2^(7-Epj%)*Sgn(Val("&X"+D.str$(Epj%))...
                                ...And 2^(11-Epi%)))
  '                          ! Byte-Wert für eine Nadelreihe bilden
```

```
    ' Hier ist wichtig, zu wissen, daß diese Byte-Werte
    ' spaltenweise von unten nach oben gebildet werden.
    ' Bit 0 des ersten Wertes liegt also im Raster unten links.
    Next Epj%
    Epdat$=Epdat$+Str$(Sp%)+"," ! Byte in DATA-Zeile einbinden
  Next Epi%
  Epdat$=Epdat$+Chr$(34)+Space$(50-Len(Epdat$))+"Chr:"
  Epdat$=Epdat$+Chr$(Ascii%)+" / Asc:"+Str$(Ascii%)+Chr$(34)
  ' DATA-Zeile formatieren und kommentieren
  Open "O",#99,"EPS_DATA.LST" ! EPS_DATA.LST öffnen
  Print #99,Epdat$;Chr$(13)   ! DATA-Zeile schreiben
  Close #99                   ! Datei schließen
```

Die zweite Routine liest nun die in den darunter stehenden DATA-Zeilen abgelegten Druckerzeichen-Definitionen, sendet die entsprechenden Initialisierungs-Sequenzen an den Drucker und daran anschließend die gelesenen Daten der Zeichen-Bit-Muster. Dabei wird gleichzeitig der User-Defined-Zeichensatz angeschaltet.

Die wohl *gebräuchlichsten selbstdefinierten* Zeichen habe ich bereits editiert und in die Prozedur eingebunden. Diese DATA-Liste können Sie auf die oben beschriebene Art und Weise beliebig erweitern oder ändern, solange als letztes DATA-Element der Wert 256 abgelegt wird.

Wenn Sie wieder den Original-Zeichensatz einsetzen möchten, senden Sie folgende Zeile an den Drucker:

```
LPRINT CHR$(27);"7";CHR$(27);"%";CHR$(0);
```

Damit werden die ASCIIs im Bereich von 128 bis 159 nicht mehr als Druckzeichen-ASCII gewertet (CHR$(27);"7";) und vom selbstdefinierten RAM-Zeichensatz in den internen ROM-Zeichensatz umgeschaltet (CHR$(27);"%";CHR$(0);).

```
@Eps_init                      ! Prozedur-Aufruf
Procedure Eps_init
  Local Ascii%,Prn.dat%,Epi%,Dat.info$
  Lprint Chr$(27);":";Chr$(0);Chr$(0);Chr$(0); ! ROM-Satz...
  '                            !        ...ins User-RAM kopieren
  Lprint Chr$(27);"%";Chr$(1); ! User-defined-Satz einschalten
  Lprint Chr$(27);"6";         ! ASCIIs 128-159 sind Druckzeichen!
  Restore Chardat              ! DATA-Label setzen
  Do                           ! Lese-Schleife
    Read Ascii%                ! ASCII lesen (erstes DATA der Zeile)
    Exit if Ascii%>255         ! Exit, wenn größer 255
    Lprint Chr$(27);"&";Chr$(0);Chr$(Ascii%);Chr$(Ascii%);
    ' Empfangs-Sequenz für selbstdefinierte Zeichen senden
    Lprint Chr$(139);          ! Unterlängen aus (nur Epson?)
    ' Bei diesem Wert (139) weiß ich nicht genau, ob er bei
    ' allen Druckern anzugeben ist. Bei Epson FX-80, FX-85 und
    ' FX-105 wird hierdurch dem Drucker mitgeteilt, daß nur
    ' ein 8-Nadel-Zeichen ohne Unterlängen zu erwarten ist.
```

```
      ' Sollte diese Zeile bei Ihrem Drucker nicht funktionieren,
      ' lassen Sie sie weg und beobachten, was passiert. Evtl.
      ' müssen Sie in Ihrem Druckerhandbuch nachschlagen, welchen
      ' Code Ihr Gerät hier erwartet bzw. ob er nötig ist.
      For Epi%=0 To 10         ! 11 Byte-Werte (Bit-Musterspalten)
        Read Prn.dat%          ! Bit-Muster-Byte lesen
        Lprint Chr$(Prn.dat%); ! Bit-Muster-Daten an Drucker
      Next Epi%
      Read Dat.info$           ! Dummy-READ für Kommentar-String
    Loop
    Chardat:
    Data 35,0,40,40,254,40,40,40,254,40,40,0,"      Chr: #/Asc: 35"
    Data 91,0,0,254,254,130,130,130,130,0,0,0,"   Chr: [/Asc: 91"
    Data 92,0,0,192,224,48,24,12,7,3,0,0,"        Chr: \/Asc: 92"
    Data 93,0,0,0,130,130,130,130,254,254,0,0,"   Chr: ]/Asc: 93"
    Data 123,0,16,16,124,238,130,130,130,0,0,0," Chr: {/Asc: 123"
    Data 124,0,0,0,0,254,254,0,0,0,0,0,"         Chr: ¦/Asc: 124"
    Data 125,0,0,0,130,130,130,238,124,16,16,0," Chr: }/Asc: 125"
    Data 129,0,0,60,128,2,0,2,128,60,2,0,"       Chr: ü/Asc: 129"
    Data 132,0,4,10,160,10,32,10,160,28,2,0,"    Chr: ä/Asc: 132"
    Data 142,0,6,140,20,36,68,36,20,140,6,0,"    Chr: Ä/Asc: 142"
    Data 148,0,0,28,162,0,34,0,162,28,0,0,"      Chr: ö/Asc: 148"
    Data 154,0,60,130,0,2,0,2,0,130,60,0,"       Chr: Ü/Asc: 154"
    Data 153,0,28,162,0,34,0,34,0,162,28,0,"     Chr: Ö/Asc: 153"
    Data 158,0,126,128,0,128,18,128,18,108,0,0," Chr: ß/Asc: 158"
    Data 256, <- Wert größer 255 = Endmarkierung für Leseschleife
  Return
```

5.8 Sound-Chip-Anweisung

SOUND {SO} — Klangausgabe

SOUND Kanal,Volume,Note,Oktave,Dauer
SOUND Kanal,Volume,#Periode,Dauer

Mit diesem Befehl können die drei Sound-Register des ST aktiviert werden. Die fünf Parameter der ersten Syntax-Variante haben folgende Bedeutung:

Kanal	Soundregisternummer (1, 2 oder 3).
Volume	Lautstärke (0 = Aus/15 = Laut).
Note	Oktave (1 - 8).
Oktave	Note (1 - 12).
Dauer	Haltedauer des Tones in 1/50 Sekunden. Das Programm wird erst nach Ablauf dieser Zeitspanne fortgesetzt.

Chromatische Tonhöhen-Tabelle (Note):

```
1  2   3  4   5  6  7   8  9   10 11 12
-----------------------------------------
C  Cis D  Dis E  F  Fis G  Gis A  B  H
```

Beispiel 1 (Kammerton A):

```
Sound 1,15,10,4,100
```

Statt der beiden Parameter Note und Oktave kann der Parameter Periode übergeben werden, dem dann das Nummernzeichen # voranzustellen ist. Der Wert dieses Parameters errechnet sich aus der physikalischen Schwingungsfrequenz eines Tones wie folgt:

```
Periode=TRUNC(125000/Frequenz+0.5)
```

Beispiel 2 (Kammerton A = 440 Hertz):

```
Sound 1,15,#284,100    ! TRUNC(125000/440+0.5)=284
```

Beispiel 3:

```
For I%=30 To 2000 Step 10
  Sound 1,15,#I%,2
Next I%
```

Höhere Periode-Werte als 4100 werden durch Periode MOD 4100 zurückgerechnet.

Um eine mehrstimmige Klangausgabe zu erreichen, sind vorher durch WAVE die Ausgabe-Attribute einzurichten, anderenfalls ist nur eine monophone Klangausgabe möglich. Hintenstehende Parameter, die vom vorherigen SOUND übernommen werden sollen, können entfallen.

WAVE { WA } Sound-Attribute einstellen

WAVE Kanal,Hüll,Form,Länge,Dauer

Mit WAVE können die für die Sound-Gestaltung zuständigen Register im Yamaha-Sound-Chip angesprochen werden. Es sind fünf Parameter zu übergeben:

Kanal

Sound- und Rauschregister an/aus. Es existieren drei Register für die Tonerzeugung und drei für die Geräuscherzeugung. Der Wert diese ersten Parameters errechnet sich aus der Bit-Stellung eines 6-Bit-Wertes.

Die Bits 0 bis 2 dieses Wertes sind der Reihe nach für die drei Sound-Generatoren zuständig. Die Bits 3 bis 5 stehen dagegen für die drei Rausch-Generatoren. Je nachdem, welches Bit gesetzt ist, wird der entsprechende Generator ein- oder ausgeschaltet, z.B. bedeutet &X101011 (bzw. 43):

```
Sound auf Kanal 1 an
Sound auf Kanal 2 an
Rauschen auf Kanal 1 an
Rauschen auf Kanal 3 an
```

Hüll

Nach dem gleichen 6-Bit-Schema wie unter Kanal beschrieben, wird bestimmt, welchem Register die aktuelle - in Kurvenform angegebene - Hüllkurve zugeordnet werden soll.

Form

Bestimmung der Hüllkurvenform. Hier sind nur die Werte 8 bis 15 interessant:

```
 8 =  \¦\¦\¦\¦\¦\  = Sägezahn/anfangs fallend
 9 =  \__________  = Linear/anfangs fallend
10 =  \/\/\/\/\/\  = Dreieck/anfangs fallend
11 =  \¦---------  = linear/sprunghaft/laut haltend
12 =  /¦/¦/¦/¦/¦/  = Sägezahn/anfangs steigend
13 =  /----------  = steigend/laut haltend
14 =  /\/\/\/\/\/  = Dreieck/anfangs steigend
15 =  /¦_________  = Attack/Cut (kurzer Anschlag)
```

Länge

Dehnung der Hüllkurve. Je größer der Wert, umso größer der Kurven-Intervall (die Modulationsfrequenz).

Dauer

Haltedauer bis zur Programmfortsetzung in 1/50 Sekunden.

Hintenstehende Parameter, die ggfs. vom vorherigen WAVE übernommen werden sollen, können entfallen.

WAVE 0,0 bzw. ein Tastaturklick schaltet alle Kanäle wieder ab.

Hier nun ein kleines Experimental-Programm, das Ihnen helfen soll, die fast undurchschaubare Logik der SOUND-Programmierung in den

Griff zu bekommen. Man muß dazu erwähnen, daß die Schöpfer des Yamaha-Chips YM-2149 sich nicht gerade bemüht haben, anderen den Zugang zu ihrem Wunderwerk zu erleichtern. Aber so ist das mit Genie-Streichen, sie sind für den "einfachen Menschen" oft schwer zu erfassen.

```
Sound 1,15,1,4                ! Feststehende
Sound 2,15,4,4                ! SOUND-
Sound 3,15,9,4                ! Einstellungen
Print At(28,5);"WAVE-Parameter :"
Print At(20,7);"    <F1>     -     <F5>      =  +"
Print At(20,8);"<Shift><F1> - <Shift><F5>  =  -"
Dim P(5)                      ! Parameter-Feld
Print At(28,10);"WAVE 0,0,0,0,0"
Repeat
  A=Asc(Right$(Inkey$))-58    ! Tastatur abfragen
  If A<6 And A>0              ! <F1> bis <F5>?
    Inc P(A)                  ! Parameter erhöhen
    Print At(28,10);"WAVE ";P(1) Mod 64;",";P(2) Mod 64;
    Print ",";P(3) Mod 16;",";P(4)*50;",";P(5)'''''
    Wave P(1) Mod 64,P(2) Mod 64,P(3) Mod 16,P(4)*50,P(5)
  Endif
  If A<31 And A>25            ! <Shift><F1> bis <Shift><F5>?
    P(A-25)=Max(0,P(A-25)-1) ! Parameter vermindern
    Print At(28,10);"WAVE ";P(1) Mod 64;",";P(2) Mod 64;
    Print ",";P(3) Mod 16;",";P(4)*50;",";P(5)'''''
    Wave P(1) Mod 64,P(2) Mod 64,P(3) Mod 16,P(4)*50,P(5)
  Endif
Until A=0
```

Beachten Sie zum Thema "Musik auf dem ST" auch 21.5.3 "Interrupt-Sound".

6. Programmstruktur

6.1 Schleifenkonstruktionen

Version 3.0

DO ... LOOP { DO ... L } **Endlosschleife**
DO [WHILE Bed] [UNTIL Bed]
...Doppelt bedingte Schleife **(V3.0)**
LOOP [WHILE Bed] [UNTIL Bed]

DO
... auszuführende Programmteile
LOOP

Folgende Syntaxvariante gilt nur für V3.0:

DO [WHILE Bedingung] [UNTIL Bedingung]
... auszuführende Programmteile, wenn DO-Bedingung
... wahr ist, bzw. solange LOOP-Bedingung wahr ist.
LOOP [WHILE Bedingung] [UNTIL Bedingung]

Eine DO...LOOP-Schleife der Versionen V2.xx kann nur abgebrochen werden, wenn sie auf eine Abbruch-Anweisung (END, EDIT, STOP) trifft, eine EXIT IF-Anweisung findet und die Abbruchbedingung wahr ist, eine GOTO-Anweisung innerhalb der Schleife zu einem Label außerhalb der Schleife verzweigt oder die Break-Funktion verwendet wird.

In der Version V3.0 ist es möglich, eine DO...LOOP-Konstruktion mit Ein- und Ausgangsbedingungen zu versehen. Dazu kann sowohl bei DO, als auch bei LOOP entweder eine WHILE- oder eine UNTIL-Bedingungsabfrage hinzugefügt werden (siehe WHILE...WEND, bzw. REPEAT...UNTIL). Wenn allein DO...LOOP eingesetzt wird, ist es möglich, statt LOOP auch ENDDO { ENDD } zu verwenden.

DO...LOOP kann - wie alle anderen Schleifen auch - beliebig tief verschachtelt werden.

Beachten Sie bitte das Beispiel zu EXIT IF.

FOR ... NEXT { F ... N } Zählschleife

FOR Zaehl=Start TO [DOWNTO] Ende [STEP Schritt]
... auszuführende Programmteile
NEXT Zaehl

Mit der Konstruktion FOR...NEXT wird eine indizierte Wiederholungschleife angelegt.

Die Kopfzeile der Schleife enthält den Anfangswert Start und den Endwert Ende. Die numerische Zählvariable Zaehl wird bei Schleifenbeginn mit dem Wert Start belegt und dann bei jedem Durchlauf solange erhöht, bzw. vermindert, bis sie den Wert Ende erreicht hat. Dabei werden alle Programmzeilen, die zwischen FOR und dem dazugehörigen NEXT eingeschlossen sind, bei jedem Durchlauf ausgeführt. Nach Erreichen des Ende-Wertes wird das Programm mit der auf die NEXT-Anweisung folgenden Programmzeile fortgesetzt.

Wird nur FOR..TO..NEXT ohne die Option STEP verwendet, beträgt die Schrittweite **immer** +1. Bei Verwendung von FOR..DOWNTO..NEXT ist der Anfangswert größer als der Endwert anzugeben, da in diesem Fall die Schrittweite **immer** -1 beträgt. Die Verwendung von STEP ist hier nicht zulässig. Die Option STEP (nur bei TO-Schleifen) bewirkt, daß der nach STEP angegebene Wert oder Ausdruck als Schrittweite angenommen wird. Hier sind auch negative Werte möglich, wobei sinnvoller Weise - wie bei DOWNTO - der Endwert kleiner als der Startwert zu wählen ist.

Ist bei Verwendung von STEP der Wertebereich von Start bis Ende nicht glatt durch den STEP-Wert Schritt teilbar, errechnet sich der zuletzt verarbeitete Zählwert aus:

```
Abs(Ende-Start)-((Abs(Ende-Start)) Mod Abs(Schritt))
```

Ist bei positiven Schleifen Start größer als Ende bzw. bei negativen Schleifen Ende größer als Start, so wird die Schleife trotzdem mindestens einmal mit dem Start-Wert durchlaufen.

In Version V3.0 kann statt NEXT Var auch ENDFOR Var { ENDF } angegeben werden.

Beispiel 1 (Hires/Midres):

```
Yt%=Min(2,3-Xbios(4))     ! Y-Auflösungsteiler
For I%=31 Downto 0        ! 32 Boxen in X-Richtung
  For J%=0 To 9           ! 2 mal 9 Boxen in Y-Richtung
```

```
      X%=31-I%                ! X-Index umkehren
      Y%=19-J%                ! Y-Index umkehren
      Box 4+I%*20,(4+J%*20)/Yt%,4+I%*20+12,(4+J%*20+12)/Yt%
      Box 4+I%*20,(4+Y%*20)/Yt%,4+I%*20+12,(4+Y%*20+12)/Yt%
      Box 6+X%*20,(6+J%*20)/Yt%,6+X%*20+8,(6+J%*20+8)/Yt%
      Box 6+X%*20,(6+Y%*20)/Yt%,6+X%*20+8,(6+Y%*20+8)/Yt%
    Next J%
  Next I%
```

Beispiel 2 (als Befehlserweiterung konzipiert):

```
Deyffill ,3,7
Pbox 10,10,200,90
Deffill ,2,4
Pcircle 30,30,20
@Mirror(0,1,10,10,200,90,10,10)
@Mirror(1,2,10,10,200,90,10,10)
@Mirror(1,1,10,10,200,90,10,100)
@Mirror(0,1,10,100,200,180,10,100)
Procedure Mirror(Md%,Gm%,Xl%,Yo%,Xr%,Yu%,X2%,Y2%)
  '
  ' Spiegelt einen beliebigen Bildschirmausschnitt
  '
  ' Md% = Mirror-Modus
  '       0 = Um die Horizontalachse spiegeln
  '       1 = Um die Vertikalachse spiegeln
  ' Gm% = Grafikmodus
  '       1 = Replace     ; 2 = Transparent
  '       3 = XOR         ; 4 = Invers transparent
  ' Xl%/Yo% = Quellbox-Koordinaten links oben
  ' Xr%/Yu% = Quellbox-Koordinaten rechts unten
  ' X2%/Y2% = Zielbox-Koordinaten links oben
  '
  ' Die Ausführung kann jederzeit durch Maustastendruck
  ' abgebrochen werden.
  '
  Local Cn%,M1$,M2$,Gm$,Rm%
  Gm$=Chr$(3)+Chr$(7)+Chr$(6)+Chr$(12) ! Grafikmodus-Tabelle
  Rm%=Asc(Mid$(Gm$,Min(4,Max(1,Gm%)),1)) ! Modus umformen
  If Md%=0                             ! Horizontal?
    For Cn%=0 To Int((Yu%-Yo%)/2)      ! Zeilen durchgehen
      Get Xl%,Yo%+Cn%,Xr%,Yo%+Cn%,M1$  ! Zeilen...
      Get Xl%,Yu%-Cn%,Xr%,Yu%-Cn%,M2$  ! ...aufnehmen
      Exit if Mousek                   ! Mausklick?
      Put X2%,Y2%+Cn%,M2$,Rm%          ! Zeilen...
      Put X2%,(Y2%+(Yu%-Yo%))-Cn%,M1$,Rm% ! ...ablegen
    Next Cn%                           ! Nächste Zeile
  Endif
  If Md%=1                             ! Vertikal?
    For Cn%=0 To Int((Xr%-Xl%)/2)      ! Spalten durchgehen
      Get Xl%+Cn%,Yo%,Xl%+Cn%,Yu%,M1$  ! Spalten...
      Get Xr%-Cn%,Yo%,Xr%-Cn%,Yu%,M2$  ! ...aufnehmen
      Exit if Mousek                   ! Mausklick?
      Put X2%+Cn%,Y2%,M2$,Rm%          ! Spalten...
      Put (X2%+(Xr%-Xl%))-Cn%,Y2%,M1$,Rm% ! ... ablegen
    Next Cn%                           ! Nächste Spalte
  Endif
Return
```

REPEAT ... UNTIL { REP ... U } Bedingte Schleife

REPEAT
... auszuführende Programmteile
UNTIL Bedingung

Die REPEAT...UNTIL-Schleife kann immer dann angewendet werden, wenn die Anzahl der Schleifendurchläufe nicht durch das Erreichen eines Endwertes (FOR...NEXT) festgelegt ist und die Schleife mindestens einmal durchlaufen werden soll.

Die Bedingung zum Verlassen der Schleife wird hier am Schleifenende geprüft. D.h., daß die Schleife mindestens einmal bis zu der in UNTIL vereinbarten Bedingung durchlaufen wird. Es sei denn, daß innerhalb der Schleife eine EXIT IF-Bedingung mit "wahr" beantwortet wird (siehe EXIT IF). Ist "Bedingung" wahr, wird das Programm mit der nächsten auf UNTIL folgenden Zeile fortgesetzt.

Diese Konstruktion läßt sich bestens dazu verwenden, um mehrfach kombinierte Abbruchbedingungen zu stellen. So können z.B. gleichzeitig ein bestimmter Tastatur-, Maus- und/oder Mausknopf-Status, der Inhalt von Variablen und/oder das Erreichen bestimmter Limits als Bedingung(en) angegeben werden.

Angenommen, für den Abbruch einer REPEAT...UNTIL-Schleife sollen die folgenden Bedingungen ausschlaggebend sein:

Mausklick rechts **und** Zählwert größer 100

oder <Esc>-Taste gedrückt

oder es sind mehr als 10 Sekunden vergangen

Die Schleifenkonstruktion sieht dann so aus:

```
T%=Timer                     ! Timer festhalten
Repeat                       ! Schleifenstart
  Inc A%                     ! Zähler +1
  K=Mousek                   ! Maustasten-Abfrage
  Key%=Asc(Right$(Inkey$))   ! Tastatur-Abfrage
Until (K=2 And A%>100) Or Key%=27 Or (Timer-T%)>2000
```

Mit den Booleschen Operatoren AND/OR/NOT/XOR/IMP/EQV lassen sich diese Bedingungen auf das abenteuerlichste miteinander verknüpfen. In Version V3.0 kann statt UNTIL auch ENDREPEAT {

ENDR } verwendet werden. Der Interpreter wandelt diesen Ausdruck dann selbstständig in UNTIL um.

WHILE ... WEND { W ... WE } — Bedingte Schleife

WHILE Bedingung
... auszuführende Programmteile...
WEND

Die WHILE...WEND-Schleife hat die Eigenschaft, daß die Abfrage der Abbruch-Bedingung bereits am Anfang der Schleife stattfindet.

So kann es sein, daß die Schleife zwar in der ersten Zeile betreten wird, jedoch das Programm sofort hinter dem dazugehörigen WEND fortgesetzt wird. Dann nämlich, wenn die Bedingung bereits bei Betreten der Schleife erfüllt ist. In diesem Fall werden die zwischen WHILE und WEND eingeschlossenen Programmzeilen also **nicht** ausgeführt. Beachten Sie hierzu bitte das Beipiel zu EOF().

In Version V3.0 kann statt WEND auch ENDWHILE { ENDW } verwendet werden. Der Interpreter wandelt diesen Ausdruck dann selbstständig in WEND um.

6.2 Bedingte Verzweigungen

EXIT IF { E IF } — Bedingter Schleifenabbruch

V3.0: { EX }

EXIT IF Bedingung

Ein sehr nützlicher Befehl, wenn es darum geht, innerhalb von FOR...NEXT-, DO...LOOP-, REPEAT...UNTIL- und WHILE...WEND-Schleifen an beliebigen - und beliebig vielen - Stellen zusätzliche Abbruchbedingungen stellen zu können.

Eine solche Schleife kann unabhängig von ihrem Zustand jederzeit verlassen werden, sobald darin eine durch EXIT IF gestellte Bedingung mit "wahr" beantwortet wird. Das Programm wird dann mit der auf den zugehörigen Schleifenwendepunkt folgenden Programmzeile fortgesetzt. Dabei ist zu beachten, daß das Programm direkt vom auslösenden EXIT IF hinter den Wendepunkt springt. Zwischen dem

auslösenden EXIT IF und dem Wendepunkt stehende Programmzeilen werden also **nicht** noch einmal ausgeführt.

Das Beispiel zu REPEAT...UNTIL läßt sich nun anhand einer DO...LOOP-Schleife und EXIT IF folgendermaßen umstrukturieren:

```
T%=Timer                       ! Timer festhalten
Do                             ! Schleifenstart
  Exit If (Timer-T%)>2000      ! 10 Sekunden vergangen?
  Inc A%                       ! Zähler +1
  K=Mousek                     ! Maustasten-Abfrage
  Key%=Asc(Right$(Inkey$))     ! Tastatur-Abfrage
  Exit If Key%=27              ! <Esc>-Taste gedrückt?
  Exit If K=2 And A%>100       ! Mausklick rechts und
  '                            ! Zähler größer 100?
Loop
```

Die - oben schon erwähnte - Besonderheit ist hier, daß die Schleife direkt verlassen wird, sobald 10 Sekunden vergangen sind und bis dahin keine der beiden anderen Abbruchbedingungen erfüllt wurde. In diesem Fall heißt das, daß der **vor** dem auslösenden Timer-Abbruch gültige Maustasten- und Tastaturstatus, sowie der Zähler A% erhalten bleiben und ggfs. weiterverwertet werden können.

In Version V3.0 kann EXIT IF auch innerhalb von IF...ENDIF- oder SELECT...ENDESELECT-Blöcken eingesetzt werden. Das Programm wird dann ggfs. direkt hinter ENDIF bzw. ENDSELECT fortgesetzt.

IF [ELSE] ENDIF { I... [EL...] EN } **Bedingungsabfrage**

In V3.0: { I... [E...] EN }

Version 3.0

ELSE IF { E IF } **Unter-Bedingungsabfrage**

IF Bedingung [THEN]
... auszuführende Programmteile,
wenn Bedingung wahr ist
[ELSE
... auszuführende Programmteile,
wenn Bedingung unwahr ist]
ENDIF

Folgende Syntaxvariante gilt nur für V3.0:

IF Bedingung1 [THEN]
... auszuführende Programmteile,
wenn Bedingung1 wahr ist
[ELSE IF Bedingung2
... auszuführende Programmteile, wenn Bedingung1
unwahr und Bedingung2 wahr ist.]
[ELSE IF Bedingung3
... auszuführende Programmteile, wenn alle
vorherigen Bedingungen unwahr waren,
jedoch Bedingung3 wahr ist.]
...
... ggfs. weitere ELSE IF-Abfragen
[ELSE
... auszuführende Programmteile, wenn alle
vorherigen Bedingungen unwahr waren.]
ENDIF

In GFA-BASIC ist es möglich, die Ausführung auch umfangreicher Programmteile allein von der Erfüllung einer einzigen Bedingung abhängig zu machen. Die Folge an Befehlen, die abhängig von einer IF-Bedingung ausgeführt werden sollen, wird hier nur von dem zugehörigen ENDIF (bzw. ELSE) eingegrenzt.

Ist Bedingung wahr, werden die zwischen IF und dem zugehörigen ENDIF bzw. ELSE stehenden Programmteile ausgeführt. Bei Verwendung der Option ELSE werden die zwischen ELSE und dem zugehörigen ENDIF eingeschlossenen Befehle ausgeführt, wenn die Bedingung sich als unwahr erweist. Ist ein IF- oder ELSE-Block ausgeführt, wird das Programm mit der auf das zugehörige ENDIF folgenden Programmzeile fortgesetzt.

IF-Abfragen können beliebig tief verschachtelt werden. Der optionale Zusatz THEN hinter IF ist zur Kompatibilität mit anderen BASIC-Dialekten gedacht. Er kann in GFA-BASIC vernachlässigt werden.

Ab hier nur für V3.0!: Durch den V3.0-Zusatzbefehl ELSE IF ist es möglich, sich Verschachtelungen folgender Art zu ersparen:

```
If Bedingung1
... Programmblock 1      >--------.
Else                               |
  If Bedingung2                    |
  ... Programmblock 2    >------.  |
  Else                          |  |
    If Bedingung3               |  |
    ... Programmblock 3  >----. |  |
    Else                      | |  |
```

```
      ... Programmblock 4  >--. | | |
      Endif                   | | | |
    .<------------------------.-' | |
    Endif                         | |
  .<------------------------------' |
  Endif                             |
  .<--------------------------------'
```

Mit ELSE IF sieht dieselbe Struktur so aus:

```
If Bedingung1
... Programmblock 1  >--------.
Else If Bedingung2            |
... Programmblock 2  >------. |
Else If Bedingung3          | |
... Programmblock 3  >----. | |
Else                      | | |
... Programmblock 4  >--. | | |
Endif                   | | | |
.<----------------------.-.-.-'
```

Ist hier die Eingangs-IF-Bedingung unwahr und trifft das Programm auf eine ELSE IF-Abfrage, deren Bedingung wahr ist, wird nur der darunter angegebene Programmblock abgearbeitet und nach dessen Ausführung zu der auf das zugehörige ENDIF folgenden Programmzeile gesprungen. Wird als Abschluß die Option ELSE verwendet und keine der vorangegangenen ELSE IF-Bedingungen wurde mit wahr beantwortet, wird auch hier die unter ELSE angegebene Programmfolge alternativ zu allen Vorbedingungen ausgeführt.

Bei folgender Abfrage werden mehrere Bedingungen durch AND so miteinander verknüpft, daß der dahinterstehende Programmblock nur dann ausgeführt wird, wenn **alle** Bedingungen der Abfrage erfüllt sind. Statt des oben verwendeten Ausdrucks Bedingung setze ich hier den Ausdruck Case ein, der jedoch nichts mit der CASE-Abfrage zu tun hat.

```
If Case1=True And Case2=True And Case3=False
  ...
  auszuführender Programmblock
  ...
Endif
```

Eine solche Struktur ließe sich mit derselben Wirkung auch in mehrere IF-Abfragen auflösen:

```
.---    If Case1=True
|         '----------------.
| .---      If Case2=True <---'
| |          '-----------------.
| | .---       If Case3=False <---'
| | |            ...
| | |            auszuführender Programmblock >---.
| | |            ...                               |
| | |          Endif                               |
| | '------>.<---------------------------------'
| |           |
| |         Endif
| '------>.
|           |
|         Endif
'------->
```

Auch hier wird der Block nur ausgeführt, wenn **alle** Abfragen wahr sind. Auf den ersten Blick mag die zweite Version etwas aufweniger erscheinen. Sie hat jedoch den Vorteil, daß nach jeder Einzel-Abfrage auf die Erfüllung der einzelnen Bedingung reagiert werden kann.

```
Fileselect "\*.DAT",".DAT",F$
If F$>""                              ! Case1
  Print "Okay-Box oder Doppelklick"   ! Reaktion
  If Right$(F$,4)=".DAT"              ! Case2
    Print "Korrekte Extension         ! Reaktion
    If Len(F$)>4                      ! Case3
      Print "Dateiname ";Left$(F$,Len(F$)-4) ! Block..
      Print "Extension = .DAT"        ! ...ausführen
    Endif
  Endif
Endif
```

Ähnliches läßt sich nun auch mit mehreren ELSE IFs anstatt einer OR-Kette realisieren. Bei der OR-Kette wird die Abfrage passiert und der Programmblock ausgeführt, sobald nur **eine** Bedingung der Kette wahr ist. Hier wurden - um einen Vergleich mit der oben beschriebenen AND-Kette zu ermöglichen - die Bedingungen negiert.

```
If Case1<>True Or Case2<>True Or Case3<>False
  Reaktion auf Negation der Bedingungen
Else
  ...
  auszuführender Programmblock
  ...
Endif
```

Mit der ELSE IF-Struktur kann jede der drei OR-Bedingungen einzeln abgefragt werden und dann auf die Erfüllung der jeweiligen Bedingung separat reagiert werden:

```
.----< If Case1<>True
!        Reaktion auf Case1        >--------------.
.<===> Else If Case2<>True                         !
!        Reaktion auf Case2        >------------. !
.<===> Else If Case3<>False                     ! !
!        Reaktion auf Case3        >----------. ! !
!      Else                                   ! ! !
'----->   .. auszuführender Programmblock >--. ! ! !
       Endif                                 ! ! ! !
        <------------------------------------'-'-'-'
```

In der Praxis könnte das dann so aussehen:

```
Fileselect "\*.DAT",".DAT",F$
If F$=""                                  ! Case1
  Print "Abbruch-Box angeklickt"          ! Reaktion
Else If Right$(F$,4)<>".DAT"              ! Case2
  Print "Falsche Extension (kein .DAT)    ! Reaktion
Else If Len(F$)=4                         ! Case3
  Print "Kein vollständiger Dateiname "   ! Reaktion
Else              ! Keine der Bedingungen ist wahr!
  Print "Korrekte Dateiauswahl!"          ! Block...
Endif                                     !...ausführen
```

Der wesentliche Unterschied zwischen der Normal-IF-Struktur und der ELSE IF-Struktur ist, daß beim Normal-IF nach jedem Programmblock die ENDIF's der vorangegangenen Abfragen noch passiert werden. Daraus ergibt sich die Möglichkeit, abhängig von dem ausgeführten Block weitere Programmteile ausführen zu lassen, die mit den vorangegangenen Abfragen in Bezug stehen. Die Notwendigkeit zu Raffinessen dieser Art ist zwar äußerst gering, kann jedoch nicht absolut ausgeschlossen werden.

Beispiel:

```
A%=3
If A%=1
  Print 1'      >----------.
Else                       !
  If A%=2                  !  Obwohl hier weder die
    Print 2'    >--------. !  erste, noch die zweite
  Else                   ! !  Bedingung zutrifft, werden
    If A%=3              ! !  die vor ihren ENDIF's
      Print 3'  >------. ! !  stehenden Zeilen Print 5
    Else               ! ! !  und Print 6 ausgeführt.
      Print 4'  >----. ! ! !  Wäre A% = 2, würde die
    Endif            ! ! ! !  Zeile Print 5 nicht mehr
    .<---------------'-' ! !  ausgeführt werden. Bei A% = 0
    Print 5'             ! !  würden außer der letzten
  Endif                  ! !  ELSE-Anweisung Print 4
  .<---------------------' !  ebenfalls noch Print 5 und
  Print 6'                 !  Print 6 ausgeführt werden.
Endif                      !
.<-------------------------'
```

Diese Möglichkeit ist bei der ELSE IF-Struktur nicht gegeben, da dort einfach kein Platz für die Zeilen Print 5 und Print 6 wäre.

Gemeinsam ist beiden Strukturen, daß immer dann, wenn eine Bedingung zutrifft, die nachfolgenden Abfragen übersprungen, also ausgeschlossen werden.

Bei IF-Verschachtelungen:

```
If Bedingung1
  ...          >---.
  .<---------------'
  If Bedingung2
  ...          >---.
  .<---------------'
  Endif
Endif
```

oder IF-Verkettungen:

```
If Bedingung1
  ...          >----.
Endif               |
.<------------------'
If Bedingung1
  ...          >----.
Endif               |
.<------------------'
```

ist dieser Ausschluß nicht gewährleistet, da mit einem Fall auch mehrere Bedingungen gleichzeitig erfüllt sein können.

Sie werden sich evtl. fragen, wie denn überhaupt die Entscheidung wahr oder unwahr getroffen wird. Bei einfachen Abfragen, wie z.B. If A%=1, ist das leicht zu erklären. Hat die Variable A% den Wert 1, wird die Bedingung mit wahr (TRUE = -1) beantwortet, andernfalls mit unwahr (FALSE = 0).

Geben Sie bitte im Direktmodus ein:

```
Print 1=1
```

Der Interpreter vergleicht nach dem Booleschen Verfahren "logisch" die beiden gegenübergestellten Werte 1 und 1. Anschließend liefert er Ihnen den Wahrheitswert -1 (TRUE). Wie das Ganze maschinenintern durch sogenannte Gatter geregelt wird, soll uns hier nicht interessieren. Wichtig ist, daß Sie wissen, **daß** es so ist. Schauen Sie sich bitte

dazu die Bedeutung der einzelnen Operatoren in Kapitel 8.1, sowie die Vorrang-Regelung der Operatoren untereinander an.

Bei komplizierteren Verknüpfungen gibt es schon einige Probleme. Hier ist es wichtig, auf eine exakte und logisch richtige Klammersetzung zu achten. So können sich aus (oberflächlich gesehen) immer demselben Ausdruck mehrere Resultate ergeben:

```
Print (2^2=4)+((7-1)>3)    -> Ausgabe:  -2
Print 2^2=4+((7-1)>3)      -> Ausgabe:   0
Print (2^2=4)+(7-1)>3      -> Ausgabe:  -1
Print (2^2=4)+7-(1>3)      -> Ausgabe:   6
```

Verblüffend, oder? - Die Klammersetzung macht es möglich!

Nun die Erklärung: In der ersten Zeile wird der Ausdruck (2^2=4) zusammengefaßt. Dasselbe geschieht mit ((7-1)>3). Beide Ausdrücke werden - jeder für sich - auf ihren Wahrheitsgehalt geprüft. Der erste Ausdruck ergibt TRUE (-1), da die zweite Potenz von 2 tatsächlich 4 ist. Der zweite Ausdruck liefert ebenfalls TRUE (-1), da 7 minus 1 (also 6) tatsächlich größer als 3 ist. Durch das Plus-Zeichen werden nun beide Wahrheitswerte addiert. Und das ergibt (-1)+(-1)= -2.

In der zweiten Zeile habe ich einfach die Klammer um den ersten Ausdruck weggelassen. Daraus ergibt sich eine völlig andere Konstellation. In diesem Fall wird zuerst das Ergebnis von 2^2 berechnet (4). Das Gleichheitszeichen wird hier allerdings nicht als zum ersten Ausdruck zugehörig erkannt, sondern stellt den Verknüpfungs-Operator zum zweiten Ausdruck dar. Dieser liefert - wie in der ersten Zeile - wieder TRUE (-1). In der endgültigen Auswertung ergibt sich also 4=(-1). Da 4 nun aber nicht -1 ist, liefert die Zeile den Unwahrheitswert FALSE, also 0.

In der dritten Zeile haben wir als ersten Ausdruck wieder (2^2=4). Hier ist die Änderung beim zweiten Ausdruck zu finden. Es fehlt die äußere Klammer. 2 hoch 2 ist 4 - stimmt! Also erhalten wir aus dem ersten Ausdruck wieder TRUE (-1). Der zweite Ausdruck ist in diesem Fall jedoch auf (7-1) reduziert. Das ergibt 6. Als nächstes werden die beiden Ausdrücke addiert. 6+(-1) ist 5. Dieses Additionsergebnis wird nun abschließend daraufhin geprüft, ob es größer als 3 ist. Da 5 tatsächlich größer als 3 ist, erhalten wir als Gesamtergebnis dieser Zeile ein TRUE (-1).

Die vierte und letzte Zeile hat immer noch - oberflächlich gesehen - eine frappierende Ähnlichkeit mit ihren Vorgängerinnen. Hier habe

ich einfach die 7 aus dem zweiten Ausdruck ausgeklammert und (1>3) zusammengefaßt. Der erste Ausdruck ist uns schon bekannt. Er liefert - wie in Zeile 1 und Zeile 3 - den Wert -1 (TRUE). Da die Klammersetzung absoluten Vorrang vor allen anderen Operatoren hat, berechnen wir als nächstes den Ausdruck (1>3). Sehr wahrscheinlich wissen Sie so gut wie ich, daß 1 **nicht** größer als 3 ist. Also erhalten wir den Wert 0 (FALSE). Mit etwas mathematischer Vorkenntnis lassen sich die drei Faktoren nun folgendermaßen zusammenfassen: (-1)+7-(0) = 6.

So weit, so gut - wenn da nicht noch die logischen Operatoren wären.

```
Print 10*10 And 14 Or 3<>2       -> Ausgabe:  -1
Print 10*10 And (14 Or 3)<>2     -> Ausgabe: 100
Print 10*(10 And 14 Or 3<>2)     -> Ausgabe: -10
```

Drei neue Zeilen, die - für den Laien - fast identisch sind, und trotzdem verschiedene Ergebnisse liefern.

Des Rätsels Lösung: In Zeile 1 finden Sie keine Klammer. Daher eignet sie sich besonders dazu, die Prioritäten der verwendeten Operatoren zu erläutern. Als erstes wird in diesem Fall die Punkt-Rechnung durchgeführt, also 10*10. Das ergibt 100. Als nächstes kommt die Vergleichsoperation <> dran. Alle Vergleiche haben Vorrang vor logischen Operatoren. Es soll hier festgestellt werden, ob 3 ungleich 2 ist. Da es das ist, erhalten wir hieraus den Wert -1 (TRUE). Zum Schluß bearbeiten wir in dieser Zeile die Booleschen Operatoren. Da beide auf derselben Ebene liegen, ergibt sich die Frage, welche Operation von beiden zuerst dran kommt - AND oder OR? In solchen Fällen gilt die goldenen Regel: immer von links nach rechts! Ich setze nun voraus, daß Sie sich über das Verfahren der logischen Verknüpfungen im Kapitel 4 "Basis-BASIC" eingehend informiert haben.

```
       100 -> Binär:    01100100
AND     14 -> Binär:    00001110
                        --------
                        00000100  -> Dezimal: 4  >--.
 .---------------------------------------------------'
 '->     4 -> Binär:         100
OR      -1 -> Binär:  1...111111
                      ----------
                      1...111111  -> Dezimal: -1
                                     ============
```

Als Ergebnis der ersten Zeile erhalten wir den Wert -1 (TRUE).

In Zeile 2 habe ich hier nichts weiter gemacht, als (14 Or 3) zu einem Ausdruck zusammenzufassen. Da - wie gesagt - die Klammerrechnung Vorrang hat, erhalten wir daraus:

```
         14 -> Binär:   00001110
OR        3 -> Binär:   00000011
                        --------
                            1111  -> Dezimal: 15
```

Wie in Zeile 1 erhalten wir auch hier aus 10*10 den Wert 100. Weil auch hier wieder der Vergleich <> Vorrang vor AND hat, beziehen wir nun daraus den Wert -1 (TRUE), da der Klammerwert 15 tatsächlich ungleich 2 ist. Zum Schluß verknüpfen wir die beiden Teilergebnisse im AND-Modus:

```
         100 -> Binär:   01100100
AND       -1 -> Binär: 1..1111111
                         --------
                         01100100  -> Dezimal: 100
                                      ==============
```

Das Ergebnis der zweiten Zeile heißt 100.

Die dritte Zeile ist wiederum nur geringfügig variiert. Wie gesagt, wird erst einmal die Klammer berechnet. Darin finden wir als höchste Priorität den Ungleich-Vergleich <>. Ist drei ungleich zwei? Ja, also erhalten wir TRUE (-1). Als zweites stehen sich eine AND- und eine OR-Verknüpfung gleichwertig gegenüber. Ich fange links an:

```
          10 -> Binär:   00001010
AND       14 -> Binär:   00001110
                         --------
                         00001010  -> Dezimal: 10  >-.
   .-----------------------------------------------'
   '->    10 -> Binär:       1010
OR        -1 -> Binär: 1...111111
                       ----------
                       1...111111  -> Dezimal: -1
```

Als Ergebnis des Klammerausdrucks erhalten wir wieder -1 (TRUE). Diese -1 wird nun mit dem Wert 10 multipliziert und siehe da - es ergibt sich der Negativ-Wert -10.

Ich hoffe, daß es mir anhand dieser zwei kleinen Beispiele gelungen ist, Ihr kritisches Auge im Umgang mit Verknüpfungen dieser Art etwas geschärft zu haben. Nun wissen Sie wenigstens ungefähr, was Ihnen an Verwirrungen bevorstehen kann und über Bedingungen wie die folgende wundern Sie sich dann hoffentlich nicht mehr all zu sehr.

```
If (10*(10 And 14 Or 3<>2))=-10
  Print "Heureka!"
Endif
```

Als kleiner Leckerbissen folgt nun ein kleines Programm, das Ihnen als Anregung für die Lösung sogenannter Kniffel-Aufgaben dienen soll.

Stellen Sie sich bitte vor, Sie hätten folgende Aufgabe zu lösen:

```
     XX * X       Eine zweistellige Zahl ist mit einer
     ------       einstelligen Zahl zu multiplizieren.
      XX          Das Ergebnis muß wieder zweistellig sein.
   +  XX          Dazu wird ein zweistelliger Wert addiert
   ---------      und das Ergebnis der Rechnung muß wieder
   =  XX          zweistellig sein.
```

Der Witz an der Sache ist nun, daß für alle neun möglichen Stellen alle neun Ziffern von 1 bis 9 je einmal verwendet werden **müssen**. Es darf also jede Ziffer nur einmal vorkommen. Es ist nur ein einziges korrektes Ergebnis möglich.

Das nun folgende Programm löst diese Aufgabe völlig "unmathematisch" und es sind natürlich wesentlich elegantere mathematische Lösungen vorstellbar. Da diese jedoch extrem kompliziert sind und von Ihnen nicht erwartet werden kann, sich in die höhere Mathematik zu vertiefen, habe ich die "logische" Lösung gewählt. Mathematiker mögen mir verzeihen. Ich habe es mit der Algebra versucht, mußte jedoch nach mehreren erfolglosen Ansätzen aufgeben.

Ein weiterer Effekt dieses Programms ist der, daß daran hervorragend der zeitliche Aufwand verschiedener Konstellationen nachvollzogen werden kann. Versuchen Sie dazu einmal, daß Programm so zu variieren, daß trotz der Änderung der Algorithmus logisch erhalten bleibt. Beispielsweise kann die vorletzte IF-Abfrage:

```
If Len(Ergebnis$+B$)=9
```

ohne weiteres um drei Zeilen - direkt unter B$=Str$(B%) - versetzt werden. Außerdem könnte man die String-Umwandlung I$=Str$(I%) streichen und den Ausdruck Str$(I%) statt I$ direkt einsetzen. Oder Sie verwenden statt der 4-Byte-Integer I%, J%, K%, A%, B% und C% einfach Realvariablen. Der Zeitaufwand wird sich in allen drei Fällen - in V2.xx zum Teil wesentlich - erhöhen. In der Wahl der Varianten lassen Sie bitte Ihre Phantasie spielen. Das Wissen um den jeweiligen Zeitaufwand wird Ihnen später bei der Entwicklung und Optimierung eigener Programme sehr von Nutzen sein.

```
T%=Timer                       ! Start-Timer festhalten
For I%=12 To 98                ! Zählschleife für ersten
 '                             ! zweistelligen Wert
 If (I% Mod 11)>0 And (I% Mod 10)>0 ! Ist I% durch 11 oder 10
  ' glatt teilbar, so kann es sich nur um Schnapszahlen
  ' (22, 33 etc.) oder Nullzahlen (20, 30 etc.) handeln und
  ' die können wir hier nicht gebrauchen.
  I$=Str$(I%)                  ! I% in String umwandeln
  I2$=Str$(I% Mod 10)          ! Einerstelle von I%
  I3$=Str$(I% Div 10)          ! Zehnerstelle von I%
  For J%=1 To 9                ! Einstellige Schleife
   J$=Str$(J%)                 ! J% in String umwandeln
   If Instr(I$,J$)=0           ! Ist Ziffer J% in den
    '                          ! I%-Ziffern enthalten?
    A%=I%*J%                   ! Nein, dann multiplizieren.
    If A%>11 And A%<99         ! Ergebnis zweistellig und
     If (A% Mod 11>0) And (A% Mod 10>0) ! glatt durch 11 oder
      '                        ! 10 teilbar? (s.o.)
      Clr C%                   ! INSTR-Positionsspeicher klar
      A$=Str$(A%)              ! A% in String umwandeln
      A2$=Str$(A% Mod 10)      ! Einerstelle von A%
      A3$=Str$(A% Div 10)      ! Zehnerstelle von A%
      C%=Instr(A$,I3$)         ! Zehnerstelle von I% in A%?
      C%=Max(C%,Instr(A$,I2$)) ! Einerstelle von I% in A%?
      C%=Max(C%,Instr(A$,J$))  ! J% in A% enthalten?
      If Len(I$+J$+A$)=5 And C%=0 ! Ziffernanzahl der
       ' bisherigen Rechnung = 5 und keine der bisher
       ' verwendeten Ziffern in einer anderen enthalten?
       For K%=12 To 98         ! Schleife für die zu
        '                      ! addierende Zahl
        K$=Str$(K%)            ! K% in String umwandeln
        K2$=Str$(K% Mod 10)    ! Einerstelle von K%
        K3$=Str$(K% Div 10)    ! Zehnerstelle von K%
        Ergebnis$=I$+J$+A$+K$  ! Ziffernanzahl der bisherigen
        '                      ! Rechnung feststellen
        If Len(Ergebnis$)=7    ! Anzahl = 7?
         If (K% Mod 11>0) And (K% Mod 10>0) ! ist K%
          ' durch 10 oder 11 glatt teilbar? (s.o.)
          C%=Instr(A$,K3$)     ! Zehnerstelle von K% in A%?
          C%=Max(C%,Instr(A$,K2$)) ! Einer von K% in A%?
          C%=Max(C%,Instr(I$,K3$)) ! Zehner von K% in I%?
          C%=Max(C%,Instr(I$,K2$)) ! Einer von K% in I%?
          C%=Max(C%,Instr(K$,J$))  ! J% in K% enthalten?
          If C%=0                  ! Keine der bisherigen
           ' Ziffern in einer anderen Zahl enthalten?
           B%=A%+K%            ! Multiplikationsergebnis
           '                   ! und K% addieren
           B$=Str$(B%)         ! B% in String umwandeln
           If B%>11 And B%<99  ! Ergebnis zweistellig?
            If (B% Mod 11>0) And (B% Mod 10>0) ! Ist es
             ' weder durch 10 noch 11 teilbar? (s.o.)
             If Len(Ergebnis$+B$)=9 ! Anzahl aller
              ' verwendeten Ziffern = 9?
              C%=Instr(B$,I3$) !---------. Prüfen, ob
              C%=Max(C%,Instr(B$,I2$))   ! eine der
              C%=Max(C%,Instr(B$,A3$))   ! bisherigen
              C%=Max(C%,Instr(B$,A2$))   ! Ziffern im
              C%=Max(C%,Instr(B$,K3$))   ! Endergebnis
              C%=Max(C%,Instr(B$,K2$))   ! enthalten
```

```
                C%=Max(C%,Instr(B$,J$)) !--' sind
                If C%=0                 ! Nein?
                 Print "Sek. : ";(Timer-T%)/200
                 Print
                 Print ''I%;" * ";J%    ! dann
                 Print "  ---------"    ! Lösung
                 Print "     ";A%       ! ausgeben
                 Print "  + ";K%        ! ...
                 Print "  ---------"    ! ...
                 Print "  = ";B%        ! ...
                 End                    ! und Ende !!
                Endif
               Endif    ! Für die, die nichts mit der Funktion
              Endif     ! MAX() in den INSTR-Abfragen anfangen
             Endif      ! können:
            Endif       ! Durch INSTR soll festgestellt werden,
           Endif        ! ob eine der jeweils verwendeten Ziffern
          Endif         ! schon vorher bereits verwendet wurde.
         Next K%        ! Da mehrere INSTR-Abfragen hintereinander
        Endif           ! stehen und anschließend auf Null gestestet
       Endif            ! wird, kann eine hintenstehende Abfrage den
      Endif             ! C%-Wert einer vorangegangenen mit Null
     Endif              ! überschreiben. Um evtl. schon gefundene
    Next J%             ! Positionen zu erhalten, wird sie durch
   Endif                ! MAX() in die jeweils nächste INSTR-Abfrage
  Next I%               ! hinübergerettet.
```

Version 3.0

SELECT [CONT] CASE [TO] [DEFAULT] ENDSELECT

{ S }{ CON }{ CA }{ DEFA }{ ENDS } **Fall-Entscheidung**

SELECT Expr
CASE Konstante1 [TO Konstante2 [,[...] TO [...]]]
... auszuführende Programmteile, wenn Expr gleich Konstante1, bzw. - bei Option TO - wenn Expr innerhalb des Bereichs von Konstante1 bis Konstante2 liegt.
[CONT]
[CASE Konstante1 [,Konstante2 [,Konstante3 [,...]]]
... auszuführende Programmteile, wenn Expr gleich Konstante1 oder gleich Konstante2 oder gleich Konstante3 oder ... oder ...]
... ggfs. weitere CASE-Entscheidungen
[CONT]
[DEFAULT
... auszuführende Programmteile, wenn keine der vorhergehenden Abfragen zugetroffen hat.]
ENDSELECT

Diese Fallentscheidung bietet die Möglichkeit zu einer Expr-abhängigen Verzweigung (select = engl.: auswählen/case = engl.: falls). Viele werden diese Konstruktion aus der Sprache C kennen (Switch/Case). Expr kann ein beliebiger numerischer oder alphanumerischer Ausdruck sein, dessen Ergebnis ggfs. vorher ermittelt wird. Es ist auch die Angabe von Variablen oder Konstanten möglich.

Wird ein(e) alphanumerischer Ausdruck, Konstante oder Variable in Expr verwendet, werden davon nur die ersten vier Zeichen zum Vergleich herangezogen. Diese werden dann intern in einen 4-Byte-Wert umgewandelt.

Hinter CASE wird bei Werte-SELECT in Konstante ein numerischer Wert oder ein max. vier Zeichen langer Text als Konstante oder String-Variable angegeben, der dann daraufhin überprüft wird, ob Expr ihm entspricht. Bei Werte-SELECT angegebene Strings werden auf Gültigkeit geprüft, indem der SELECT-Wert mit den ASCII-Werten der ersten vier Zeichen des Textes (sofern vorhanden) verglichen wird.

Beispiel:

```
A%=INP(2)*2^24+INP(2)*2^16+INP(2)*2^8+INP(2)
SELECT A%
CASE "abcd"
  PRINT "abcd wurde eingegeben!"
DEFAULT
  PRINT "irgendwas!"
ENDSELECT
```

In diesem Fall werden vier Tasten abgefragt (4 mal INP(2)). Der ASCII-Wert aller einzelnen Tasten wird zu einen Gesamt-4-Byte-Wert verknüpft. Auf dieselbe Art analysiert CASE bei Werte-SELECT einen angegebenen String. Es wird also ein MKL$-, MKI$- oder CHR$-String gebildet (je nach Länge des angegebenen CASE-Strings), der dann mit Expr verglichen wird.

3-Zeichen-Strings:

```
  (ASCII-Wert des 1. Zeichens) * (2^16)
+ (ASCII-Wert des 2. Zeichens) * (2^8)
+ (ASCII-Wert des 3. Zeichens)
```

2-Zeichen-Strings:

```
  (ASCII-Wert des 1. Zeichens) * (2^8)
+ (ASCII-Wert des 2. Zeichens)
  etc.
```

Bei String-SELECT ist ebenfalls nur die Angabe eines maximal vier Zeichen langen Strings hinter CASE zulässig.

Entspricht Expr der CASE-Auswahl, wird die darauffolgende Programmsequenz ausgeführt und daran anschließend direkt zu der auf das zugehörige ENDSELECT folgenden Programmzeile gesprungen. Evtl. weitere CASE-Abfragen derselben Gruppe bleiben dann also unberücksichtigt. Hier ist die SELECT...CASE-Konstruktion mit IF...ELSE IF vergleichbar. Der wesentliche Unterschied ist der, daß der Interpreter die Kontrolle über die Gültigkeit der Auswahl-Vorgaben bei CASE automatisch vornimmt. Die ELSE IF-Konstruktion:

```
A%=Random(120)
Print A%,
If A%>5 And A%<44
  Print "innerhalb"            >---------.
Else If A%<6 Or A%>43 And A%<100         |
  Print "außerhalb < 100"      >-------. |
Else If A%>100                         | |
  Print "außerhalb > 100"      >-----. | |
Else                                 | | |
  Print "100"                  >---. | | |
Endif                              | | | |
.<---------------------------------'-'-'-'
```

sieht mit SELECT...CASE folgendermaßen aus:

```
A%=Random(120)
Print A%,
Select A%
Case 6 To 43
  Print "innerhalb"            >---------.
Case  To 6,44 To 99                      |
  Print "außerhalb < 100"      >-------. |
Case 101 To                            | |
  Print "außerhalb > 100"      >-----. | |
Default                              | | |
  Print "100"                  >---. | | |
Endselect                          | | | |
.<---------------------------------'-'-'-'
```

Die Verzweigungskriterien hinter CASE sind im Vergleich zu den >-, AND- und OR-Bedingungen der ELSE IF-Struktur durchschaubarer, da sie der Formulierung von Bedingungen im normalen Sprachge-

brauch eher entsprechen. Eine weiterer Vorteil gegenüber ELSE IF ist der erhebliche Geschwindigkeitsgewinn:

```
T%=Timer
For I%=1 To 10000        ! 10.000 Durchläufe
  Select I%
  Case 20000             ! 1. Abfrage
  Case 20000             ! 2. Abfrage
  Endselect
Next I%
Print "CASE-Abfrage : ";(Timer-T%)/200;" Sek."
T%=Timer
For I%=1 To 10000        ! 10.000 Durchläufe
  If I%=20000            ! 1. Abfrage
  Else if I%=20000       ! 2. Abfrage
  Endif
Next I%
Print "ELSE IF-Abfrage : ";(Timer-T%)/200;" Sek."
```

An den Sprunglinien im obigen Struktur-Vergleich können Sie erkennen, daß wenn eine CASE-Bedingung erfüllt ist, anschließend **keine** der folgenden CASE-Bedingungen durchlaufen wird. So wird bei folgendem Beispiel nur die Zeile PRINT "A" ausgeführt, obwohl die folgende CASE-Bedingung "A" To "Z" ebenfalls zutrifft.

```
X$="A"
Select X$
Case "A"
  Print "A"    >-------.
Case "A" To "Z"        |
  Print "A bis Z"      |
Endselect              |
.<---------------------'
```

Durch die optionale Angabe von TO kann ein ganzer Bereich angegeben werden (z.B. CASE 1 TO 10/CASE "a" TO "z" oder CASE "abc" TO "xyz"), innerhalb dessen Grenzen 'Expr' liegen muß, um die zugehörige Sequenz zu durchlaufen. Wird die erste (kleinere) Bereichsgrenze vor TO oder die zweite (größere) Bereichsgrenze nach TO weggelassen, wird intern automatisch die kleinstmögliche bzw. größtmögliche Grenze angenommen. Durch Verwendung eines Kommas als Trennzeichen können auch mehrere Einzelangaben zusammengefaßt werden (z.B. CASE a,h,j,m oder CASE 1,33,7). Es ist möglich, die CASE-Bedingungsformate in einer CASE-Zeile beliebig zu vermischen (z.B. CASE TO "b","ABC" TO "XYZ",65,66,67,"Ä").

Wurden sämtliche angegebenen CASE-Anweisungen ohne wahr-Ergebnis passiert, kann am Ende des SELECT-Blocks DEFAULT eingesetzt werden, was dazu führt, daß dann der zwischen DEFAULT und

ENDSELECT liegende Programmteil ausgeführt wird (vergleichbar mit ELSE bei IF-Abfragen).

Wird direkt vor einer CASE-Anweisung am Ende einer Verzweigung die Option CONT verwendet, bewirkt dies, daß die direkt danach stehende CASE-Abfrage übersprungen wird und die dieser CASE-Abfrage unterstellte Sequenz zusätzlich zur schon ausgeführten Verzweigung ebenfalls ausgeführt wird. Nach Erledigung dieser Folgesequenz wird - sofern nicht auch diese mit CONT abgeschlossen wurde - zur ersten Zeile hinter ENDSELECT gesprungen. CONT kann ggfs. auch direkt vor DEFAULT eingesetzt werden, was bewirkt, daß - vorausgesetzt, der vor DEFAULT stehende Programmblock wurde ausgeführt - die unter DEFAULT eingefügte Alternativ-Sequenz zusätzlich abgearbeitet wird. Steht CONT nicht direkt vor CASE oder DEFAULT, wird es als CONT zur Programmfortsetzung nach einem STOP-Befehl interpretiert.

Übrigens ist es zwecklos, Programmzeilen zwischen SELECT und dem ersten CASE unterbringen zu wollen, da diese Zeilen vom BASIC nicht registriert werden.

Statt DEFAULT kann auch OTHERWISE { OT } geschrieben werden. Dieser Ausdruck wird vom Interpreter in DEFAULT umgewandelt.

6.3 Bereichsdeklaration

! Kommentar innerhalb einer Befehlszeile

Befehlszeile ! Kommentartext

Grundsätzlich hat dieses Ausrufungszeichen die gleiche Aufgabe wie der Befehl REM. Der darauf folgende Text wird als Programm-Kommentar deklariert und vom BASIC bei der Interpretation "übersehen". Sie sind dieser Form der Kommentar-DeKlaration bei den Beispielprogrammen hier im Buch schon häufiger begegnet.

Der Unterschied zwischen ! und REM besteht darin, daß durch ! der Kommentartext direkt an eine Befehlszeile angehängt werden kann. In DATA-Zeilen ist diese Abgrenzung allerdings nicht möglich, da sie hier als Text-DATA angesehen werden würde.

Beispiel:

```
Input A$        ! String einlesen
Print A$        ! String ausgeben
Edit            ! Programmende
```

Am Anfang einer Zeile verwendet, wird ! vom V3.0-Interpreter in die REM-Abkürzung umgewandelt.

DATA {D} Daten-Speicher deklarieren

DATA [Wertedatas [,["] Textdatas ["],...]]

Der Anweisung wird ggfs. eine Liste durch Kommata getrennter Werte oder Texte übergeben. Sie dient dazu, einem auftretenden READ-Befehl die entsprechende Anzahl von Daten zur Verfügung zu stellen.

Wie bei RESTORE beschrieben, kann ein DATA-Zeiger auf eine bestimmte Marke (Label) gerichtet werden. READ liest dann der Reihe nach die DATAs, die auf die angegebene Marke folgen. Wird kein RESTORE verwendet, werden vom Programmanfang aus nacheinander soviele DATAs eingelesen, wie READ-Anweisungen vorhanden sind. Werden mehr READ-Anweisungen eingesetzt als DATAs bis zum Programmende vorhanden sind, wird eine entsprechende Fehlermeldung ausgegeben.

Eine Besonderheit des GFA-BASICs ist es, daß Text-DATAs nicht zwischen An- und Abführungsstriche gesetzt werden müssen. Es sei denn, daß in Text-DATAs Kommas vorkommen. In diesem Fall würde das Text-Komma als Trenn-Komma zum nächsten String-Teil gewertet werden. Werden die An-/Abführungzeichen vernachlässigt, liest der Interpreter alle vorkommenden Zeichen (auch Leerzeichen), die zwischen den einschließenden Kommas aufgeführt sind.

Numerische READ-Anweisungen sind darauf angewiesen, auch numerische DATAs vorzufinden. Diese können dann allerdings auch in der binären, hexadezimalen oder oktalen Schreibweise angegeben sein. Liest ein Text-READ ein numerisches Zeichen, so wird es einfach als Textzeichen interpretiert.

Variablen können in den DATAs nicht übergeben werden. Kommen im Programm keine READ-Anweisungen vor, kann in DATA-Zeilen beliebiger Text stehen. Dieser Text bleibt dann - wie bei REM - vom Programm unberücksichtigt.

Schauen Sie sich hierzu bitte auch die Beispiele zu WRITE und TAB() an. Als kleiner Leckerbissen folgt nun wieder eine Befehlserweiterung,

die die simple, aber wirkungsvolle Aufgabe hat, einen beliebigen Speicherbereich in DATA-Zeilen umzuwandeln und abzuspeichern. Da es mit dem Abspeichern von DATAs noch nicht getan ist, habe ich als Ergänzung noch eine Routine hinzugefügt, die einen DATA-Block einliest und die Daten in einen ebenfalls beliebigen Speicherbereich schreibt.

Einsatzdemo für Pcode:

```
Deffill ,2,4                      !--.
Pbox 0,0,40,40                       !--- Irgendeine Grafik
Deffill ,2,2                         !
Pcircle 20,20,15                  !--'
Get 0,0,40,40,A$                  ! Grafik in 'A$' einlesen
@Pcode_v2(2,Len(A$)/2,Varptr(A$))! Inhalt von 'A$' im
'                                 ! Wordformat abspeichern
Edit
```

Im Anschluß an den oberen Programmteil mergen Sie bitte die Datei PDATAS.LST hinter das Label Pdatas_v2: (bzw. Pdatas:) an das Programmende.

Löschen Sie nun den ersten Programmteil und starten Sie den folgenden. Beim ersten Aufruf werden das Format und die DATA-Anzahl gelesen und beim zweiten Aufruf die erhaltenen Werte übergeben. Die DATAs werden nun in den vorbereiteten Puffer gelesen und als PUT-Grafik auf den Bildschirm gebracht.

Einsatzdemo für 'Pread':

```
Restore Pdatas                    ! DATA-Label setzen
@Pread_v2(*A%,*B%,0)              ! Format und Anzahl lesen
A$=Space$(B%*A%)                  ! Puffer vorbereiten
Restore Pdatas_v2                 ! DATA-Label restaurieren
@Pread_v2(A%,B%,Varptr(A$))       ! DATAs in Puffer lesen
Put 100,100,A$                    ! Grafik ausgeben
'
Procedure Pcode_v2(P.frm%,P.anz%,P.adr%)
  ' Schreibt einen beliebigen Speicherbereich als
  ' DATA-Zeilen in die Datei 'PDATAS.LST' (für V2.xx).
  ' -------------------------------------------------------
  ' P.frm% = Verarbeitungsformat
  '          1 = Byte/2 = Word/4 = Longword
  ' P.anz% = Anzahl der zu erzeugenden DATA-Werte
  '          Beachte: Dieses ist nicht die Byte-Anzahl,
  '                   sondern die Größe des Bereichs,
  '                   geteilt durch das Format P.frm%.
  ' P.adr% = Startadresse des zu konvertierenden Bereichs
  '          Bei Word- und Longword-Format wird nicht
  '          auf die Angabe einer geraden Adresse hin geprüft!
  '
```

```
' In der Datei PDATAS.LST finden Sie nach Abschluß die
' erzeugten DATA-Zeilen, die Sie nun durch Merge in den
' Programmtext einfügen können.
'
Hidem                              ! Maus ausschalten
Local I%,J%,D$                     ! Lokale Variablen
If P.frm%=1 Or P.frm%=2 Or P.frm%=4 ! Format zulässig?
  Open "O",#99,"Pdatas.lst"        ! Datei öffnen
  P.st%=Min(18,24/P.frm%)          ! DATA-Anzahl pro Zeile
  Print #99,"D ";Str$(P.frm%);",";Str$(P.anz%) ! Format und
  '                                ! Anzahl schreiben
  For I%=1 To P.anz% Step P.st% !DATA-Leseschleife
    D$="D "                        ! DATA-Kürzel an den
    '                              ! Zeilenanfang schreiben
    For J%=1 To Min(P.st%,P.anz%-(I%-1)) !Anzahl je Zeile
      If P.frm%=1                  ! Byte-Format?
        D$=D$+Str$(Peek(P.adr%+(I%-1)+(J%-1)))+"," !einbinden
      Endif
      If P.frm%=2                  ! Word-Format?
        D$=D$+Str$(Dpeek(P.adr%+(I%-1)*2+(J%-1)*2))+"," ! -"-
      Endif
      If P.frm%=4                  ! Long-Format?
        D$=D$+Str$(Lpeek(P.adr%+(I%-1)*4+(J%-1)*4))+"," ! -"-
      Endif
    Next J%                        ! Nächstes Longword
    D$=Left$(D$,Len(D$)-1)         ! Letztes Komma abschneiden
    Print #99,D$                   ! String in Datei schreiben
  Next I%                          ! Nächste Zeile
  Close #99                        ! Datei schließen
Endif
Showm                              ! Maus wieder anschalten
Return
```

Wurde die Datei PDATA.LST in das Programm gemergt, ist vor den DATA-Block ein Label zu setzen. Die erste DATA-Zeile enthält zwei Werte: der erste stellt das beim Schreiben verwendete Format und der zweite die Gesamtanzahl der erzeugten DATAs dar. Die darauf folgenden DATA-Zeilen enthalten der Reihe nach alle erzeugten Werte.

Um zuerst das beim Schreiben verwendete Format und die DATA-Anzahl zu ermitteln, werden die ersten beiden Parameter als Pointer auf zwei Variablen angegeben und als Adresse in P.adr% eine Null übergeben. Vor Aufruf ist das gewünschte DATA-Label zu restaurieren. Aus den beiden Pointer-Variablen können danach das Format und die DATA-Anzahl entnommen werden. Vor dem eigentlichen Aufruf zum DATA-Lesen ist wieder das DATA-Label zu restaurieren und die erhaltenen beiden Werte diesmal an die Prozedur zu übergeben. In diesem Fall geben Sie in P.adr% die Startadresse des Zielbereichs an. Soll die Prozedur Pread mit von vornherein feststehendem Format und Anzahl aufgerufen werden, vergessen Sie bitte nicht, daß die Prozedur trotzdem zwei DATAs (Format und Anzahl) am DATA-Blockanfang erwartet.

```
Procedure Pread_v2(P.frm%,P.anz%,P.adr%)
  ' Liest die mit Pcode_v2 erzeugten DATAs (nachdem sie
  ' durch Merge in das Programm geladen wurden) und
  ' schreibt sie in den angegebenen
  ' Speicherbereich, bzw. liefert das verwendete Format
  ' und die DATA-Anzahl (für V2.xx).
  ' ---------------------------------------------------------
  ' P.frm% = Verarbeitungsformat:
  '           1 = Byte/2 = Word/4 = Longword
  '          bzw. Pointer auf Format-Rückgabevariable,
  '          wenn P.adr% = 0
  ' P.anz% = Anzahl der zu erzeugenden DATA-Werte
  '          bzw. Pointer auf Anzahl-Rückgabevariable,
  '          wenn P.adr% = 0
  '          Achtung: Dieses ist nicht die Byte-Anzahl, sondern
  '                   die Größe des zu konvertierenden Bereichs,
  '                   geteilt durch das Format P.frm%.
  ' P.adr% = Startadresse des Bereichs, in den die gelesenen
  '          DATAs geschrieben werden sollen, bzw. 0, wenn nur
  '          Format und Anzahl geliefert werden sollen.
  '
  ' Achtung: Bei Word- und Longword-Format wird nicht
  '          auf die Angabe einer geraden Adresse geprüft!
  '
  Hidem                          ! Maus ausschalten
  Local I%,Wrt%,Anz%,Frm%        ! Lokale Variablen
  If P.adr%=0                    ! P.adr% = 0?
    Read Frm%                    ! Format lesen
    *P.frm%=Frm%                 ! In 1. Parameter zurückgeben
    Read Anz%                    ! DATA-Anzahl lesen
    *P.anz%=Anz%                 ! In 2. Parameter zurückgeben
  Else                           ! Adresse wurde angegeben
    Read Wrt%,Wrt%               ! Format und Anzahl lesen
    '                            ! (und wieder vergessen)
    For I%=1 To P.anz%           ! P.anz% DATAs
      Read Wrt%                  ! lesen
      If P.frm%=1                ! Byte-Format?
        Poke P.adr%+(I%-1),Wrt%  ! Ab P.adr% POKEn
      Endif
      If P.frm%=2                ! Word-Format?
        Dpoke P.adr%+((I%-1)*2),Wrt% ! Ab P.adr% DPOKEn
      Endif
      If P.frm%=4                ! Long-Format?
        Lpoke P.adr%+((I%-1)*4),Wrt% ! Ab P.adr% LPOKEn
      Endif
    Next I%                      ! Nächstes Longword
  Endif
  Showm                          ! Maus wieder anschalten
Return
Pdatas_v2:                       ! DATA-Label
```

Da die Parameter-Übergabe und die Auswahlbedingungen in der V3.0-Version anders gestaltet werden können und sich dadurch außerdem auch Geschwindigkeitsvorteile ergeben, folgen nun die beiden Prozeduren noch einmal für die Version V3.0. Außerdem können Ihnen die hier vorgenommenen Änderungen als Beispiel für mögliche Optimierungen in der V3.0-Version dienen.

Vor Ausführung des folgenden Programmteils ist zuerst der Pcode_v2-Aufruf oben entsprechend abzuändern und aufzurufen.

```
Restore Pdatas                      ! DATA-Label setzen
@Pread(0,A%,B%)                     ! Format und Anzahl holen
A$=Space$(Mul(B%,A%))               ! Puffer vorbereiten
Restore Pdatas                      ! DATA-Label restaurieren
@Pread(Varptr(A$),A%,B%)            ! DATAs in Puffer lesen
Put 100,100,A$                      ! Grafik ausgeben
'
Procedure Pcode(P.frm%,P.anz%,P.adr%)
  ' - für V3.0 -
  Hidem
  Local I%,J%,D$
  Select P.frm%
  Case 1,2,4                        ! Format zulässig?
    Open "O",#99,"Pdatas.lst"       ! Datei öffnen
    P.st%=Min(18,24/P.frm%)         ! DATA-Anzahl pro Zeile
    Print #99,"D ";Str$(P.frm%);",";Str$(P.anz%) ! Format
    '                               ! und Anzahl schreiben
    For I%=1 To P.anz% Step P.st% !Daten-Leseschleife
      D$="D "                       ! DATA-Kürzel an den
      '                             ! Zeilenanfang schreiben
      For J%=1 To Min(P.st%,P.anz%-(I%-1)) ! Anzahl pro Zeile
        Select P.frm%               ! P.frm% wählen
        Case 1                      ! Byte-Format?
          D$=D$+Str$(Byte{P.adr%+(I%-1)+(J%-1)})+"," ! Einbinden
        Case 2                      ! Word-Format?
          D$=D$+Str$(Card{P.adr%+(I%-1)*2+(J%-1)*2})+"," ! -"-
        Case 4                      ! Long-Format?
          D$=D$+Str$({P.adr%+(I%-1)*4+(J%-1)*4})+","     ! -"-
        Endselect
      Next J%                       ! Nächstes Longword
      D$=Left$(D$,Len(D$)-1)        ! Letztes Komma abschneiden
      Print #99,D$                  ! String in Datei schreiben
    Next I%                         ! Nächste Zeile
    Close #99                       ! Datei schließen
  Endselect
  Showm                             ! Maus wieder anschalten
Return
Procedure Pread(P.adr%,Var P.frm%,P.anz%)
  ' - für V3.0 -
  '
  ' P.frm% = Verarbeitungsformat:
  '           1 = Byte/2 = Word/4 = Longword bzw.
  '          VAR-Format-Rückgabevariable, wenn P.adr% = 0
  ' P.anz% = Anzahl der zu erzeugenden DATA-Werte bzw.
  '          VAR-Anzahl-Rückgabevariable, wenn P.adr% = 0
  '
  ' Alles andere ist mit der V2.xx-Ausführung identisch.
  ' ------------------------------------------------------------
  Hidem
  Local I%,Wrt%
  If P.adr%=0                       ! P.adr% ist 0?
    Read P.frm%,P.anz%              ! Format und Anzahl lesen
  Else                              ! Adresse ist > 0 !
  Read Wrt%,Wrt%                    ! Format und Anzahl lesen
    '                               ! (und wieder vergessen)
```

```
        For I%=1 To P.anz%              ! P.anz% DATAs
          Read Wrt%                     ! lesen
          Select P.frm%                 ! P.form wählen
          Case 1                        ! Byte-Format?
            Byte{P.adr%+(I%-1)}=Wrt%    ! Ab P.adr% schreiben
          Case 2                        ! Word-Format?
            Card{P.adr%+((I%-1)*2)}=Wrt% ! Ab P.adr% schreiben
          Case 4                        ! Long-Format?
            {P.adr%+((I%-1)*4)}=Wrt%    ! Ab P.adr% schreiben
          Endselect
        Next I%                         ! Nächstes Longword
      Endif
      Showm                             ! Maus wieder anschalten
    Return
    Pdatas:                             ! DATA-Label
```

Von einem Bekannten wurde ich einmal gefragt, ob es möglich sei, komplette Fullscreens (32000 Byte) direkt in einem Programm unterzubringen. Nach kurzer Überlegung gab ich ihm zur Antwort, daß es "selbstverständlich" über die DATA-Methode möglich ist. Da ich von Haus aus hilfsbereit bin, setzte ich mich hin und begann, ihm ein kleines Programm dafür zu schreiben. Nach ca. 1 Stunde war es soweit. Das Ergebnis bestand aus den beiden oben gezeigten Prozeduren Pcode und Pread. Als ich es meinem Bekannten stolz vorführte, wollte sich bei ihm die rechte Begeisterung nicht einstellen. Der Grund: es dauert viele lange Sekunden, bis aus den DATAs ein komplettes Bild wurde und außerdem verbraucht der DATA-Block ca. den doppelten Speicherplatz.

Aus diesen verständlichen Gründen sind die beiden Prozeduren also nur bei relativ kleinen Datenblöcken ratsam.

Welche Alternativen bieten sich nun? In V3.0 ist die Antwort sehr einfach - INLINE! Dieser Befehl eignet sich allerdings nur bis zu einer Datenmenge von 32700 Bytes pro Einheit, was in den meisten Fällen ausreichen sollte. Außerdem sind ja beliebig (?) viele INLINE-Speicher in einem Programm möglich.

In den V2.xx-Versionen gibt es nur zwei komfortable Alternativen. Die erste besteht in sogenannten "Packern", welche die sich wiederholenden Datensequenzen zu 3- oder 4-Byte-DATAs zusammenfassen. Angenommen, in dem zu speichernden Bild sind die ersten x-hundert Bytes gleich Null. Statt diese x-hundert Bytes einzeln abzuspeichern, kann man dafür als Ersatz z.B. folgenden Drei-Byte-Code einsetzen:

```
1 und 2 =  HI- und LO-Byte der Byte-Anzahl (hier: x-hundert)
Byte 3  =  Byte-Wert (hier: Null)
```

Wenn in dem zu speichernden Datenblock häufig wiederkehrende Byte-Folgen auftreten, kann dadurch - je nach Komplexität der Daten - Speicherplatz und Zeit gespart werden. Beim ST könnte man sogar leicht 4-Byte-Codes verwenden, da die überwiegende Verarbeitungbreite das Word ist. Statt die Daten byteweise zu packen, werden sie dann wordweise gepackt.

Ein derartiger Packer hat den - teils großen - Vorteil, daß der Pack-Modus geheim gehalten werden kann und so die Daten unlesbar werden. Ein wesentlicher Nachteil ist dagegen, daß bei hochkomplexen Datenstrukturen (also dann, wenn sehr wenig wiederkehrende Sequenzen auftreten) ein Packer das Vielfache an Platz und Zeit verbraucht.

GFA-BASIC liefert uns eine sehr wirkungsvolle Codierungsmethode. Es ist nämlich möglich, im Programmtext fast alle möglichen ASCII-Zeichen direkt einzugeben, die dann auch im Listing erkennbar sind. Diese Fähigkeit kann man nun dazu nutzen, Daten so zu speichern, daß

a. nicht wesentlich mehr Platz verbraucht wird, als die zu speichernden Daten selbst ausmachen (maximal ca. das 1.1-fache).

b. die Daten schnell übertragbar sind.

c. die Daten direkt im Programm untergebracht werden können.Die Anzahl der zu codierenden Datenmenge pro Einheit ist allerdings auch hier - wie bei V3.0-INLINE - auf 32700 Bytes beschränkt, wobei auch hier beliebig (?) viele Einheiten in einem Programm möglich sind. Eingeschränkt wird dieses Speicherverfahren - genau wie bei INLINE - durch den verfügbaren Speicherplatz.

Ein wesentlicher Nachteil ist, daß sich bei Blöcken, die viele 0er-, 10er- 13er- oder 34er-Bytes enthalten, das Verhältnis von Quelldatenanzahl zur Anzahl der erzeugten Bytes wesentlich verschlechtern kann. Ob sich das Verfahren lohnt, läßt sich in den meisten Fällen erst nach erfolgter Codierung feststellen. Es eignet sich also am ehesten für solche Fälle, in denen ein Packer "die Segel streichen" muß.

Der unbestreitbare Vorteil des Verfahrens ist die erstaunliche Geschwindigkeit, mit welcher die Daten vom BASIC für das Programm verfügbar gemacht werden. Außerdem ist der Datenblock im weiteren Programmverlauf beliebig oft - ohne nochmaligen Zeitaufwand - einsetzbar. Daß auch hier die Lesbarkeit der Daten für Nicht-Befugte

erschwert werden kann (je nach Codierungs-Schema), ist ein willkommener Nebeneffekt.

Die als Erweiterung konzipierte Routine erzeugt ein .LST-File, das Sie durch die Editor-Funktion Merge in Ihr Listing einfügen können. In diesem File werden die Daten als normaler String abgespeichert. Da innerhalb von listinginternen String-Definitionen keine Anführungszeichen (CHR$(34)), Carriage Returns (CHR$(13)), Line Feeds (CHR$(10)) und Nullzeichen (CHR$(0)) vorkommen dürfen, werden diese von der Routine als Ersatz-String-Variable in den Haupt-String integriert.

Da sehr wahrscheinlich Null-Byte-Ketten die am häufigsten auftretenden Fälle sein werden, ist die Routine so gestaltet, daß diese zu max. 4-Byte-Strings zusammengefaßt und gegen entsprechende Ersatz-Strings ausgetauscht werden, so daß sich das Größenverhältnis vom Quelldaten-Block zum Code-Block ggfs. wieder zu Gunsten der Codierung ändern kann. Nach dem Prinzip dieses Null-String-Packers läßt sich natürlich auch ein kompletter Total-Packer realisieren.

Jedem erzeugten String-File wird die Definition der Ersatz-Strings vorangestellt. Haben Sie in einem Programm mehrere Strings dieses Formats untergebracht, so ist die Ersatz-String-Definition nur einmal - vor Aufruf des ersten Daten-Strings - durchzuführen.

Als Bonbon finden Sie unten eine Routine Rplc, welche beliebige Strings bzw. Zeichen in einem Vorgabe-String gegen einen Ersatz-String austauscht. Die Codierungs-Routine Scode bedient sich dieser Fähigkeiten und ist daher darauf angewiesen, die Prozedur Rplc im Listing zu finden.

Einsatzdemonstration für Scode:

```
Deffill ,2,4                     !--.
Pbox 0,0,40,40                      !--- Irgendeine Grafik
Deffill ,2,2                        !
Pcircle 20,20,15                 !--'
Get 0,0,40,40,A$                 ! Grafik in A$ einlesen
Scode(Varptr(A$),Len(A$),"test","test") ! A$ codieren
'
Procedure Scode(D.sta%,D.anz%,D.var$,D.nam$)
  ' Erzeugt einen Code im GFA-Editortext-Format und speichert
  ' diesen auf Diskette bzw. Hard-Disk ab.
  ' ------------------------------------------------------------
  ' D.sta% = Startadresse des zu codierenden Speicherbereichs
  ' D.anz% = Größe des zu codierenden Bereichs in Byte
  ' D.var$ = Name der erzeugten String-Variablen
  ' D.nam$ = File-Name, unter welchem der erzeugte String
```

```
'          abgespeichert werden soll. Es sind max. 8 Zeichen
'          zugelassen. Die Extension .LST wird von der
'          Routine selbständig angehängt. Im aufrufenden
'          Programm darf keine Datei mit der Kanalnummer #99
'          geöffnet sein.
'
Open "O",#99,D.nam$+".LST"          ! Ausgabe-Datei öffnen
Print #99;"X0$=Chr$(0)";Chr$(13);"X10$=Chr$(10)";Chr$(13); !-.
Print #99;"X13$=Chr$(13)";Chr$(13);"X34$=Chr$(34)";Chr$(13); !-.
Print #99;"X02$=X0$+X0$";Chr$(13);"X03$=X0$+X02$";Chr$(13);  ! |
Print #99;"X04$=X02$+X02$"                                   !-' |
'                        Predefinition der Ersatz-Strings  <----'
Local Lco%,Em1%,Em2%,I%,Buff$,Buff2$ ! Lokale Variablen
Lco%=15                             ! Zähler für Zeilenlänge
Buff$=D.var$+"$="+Chr$(34)          ! Ausgabe-String vorbereiten
For I%=0 To D.anz%-1                ! Alle Bytes durchgehen
  Em1%=Peek(D.sta%+I%)              ! Byte lesen
  If Em1%=0 Or Em1%=10 Or Em1%=13 Or Em1%=34 ! Gelesenes Byte
    '                               ! ist 0, 10, 13 oder 34?
    If Em2%<>34                     ! Letztes Zeichen im
      '                             ! String war 0, 10 oder 13?
      Buff$=Buff$+Chr$(34)+"+X"+Str$(Em1%)+"$" ! Ersatz-String
      '                             ! in Haupt-String einbinden
      Em2%=Asc("$")                 ! Letztes Zeichen merken
    Else                            ! Byte = 34 !
      Buff$=Left$(Buff$,Len(Buff$)-1) ! String-Anfang isolieren
      Buff$=Buff$+"X"+Str$(Em1%)+"$"  ! Ersatz-String einbinden
      Em2%=Asc("$")                 ! Letztes Zeichen merken
    Endif
    If Lco%>230                     ! Zeilenlänge > 230?
      Goto Setsta                   ! Zeile abschließen
    Else                            ! Zeilenlänge <= 230 !
      Buff$=Buff$+"+"+Chr$(34)      ! Nächsten String-Teil öffnen
      Em2%=34                       ! Letztes Zeichen merken
      Add Lco%,13                   ! Längenzähler setzen
    Endif
  Else                  ! Byte kann direkt geschrieben werden !
    Buff$=Buff$+Chr$(Em1%)          ! Byte in String einbinden
    Inc Lco%                        ! Längenzähler +1
    Em2%=Em1%                       ! Byte merken
    If Lco%>230                     ! Zeilenlänge > 230?
      Setsta:
      @Ersetzer                     ! Ketten ersetzen
      Print #99;Buff$               ! String in Datei schreiben
      Buff$=D.var$+"$="+D.var$+"$+"+Chr$(34) ! Neuer String
      Em2%=34                       ! Letztes Zeichen merken
      Lco%=30                       ! String-Zähler neu setzen
    Endif
  Endif
Next I%                             ! Nächstes Byte
@Ersetzer                           ! Ketten ersetzen
Print #99;Buff$                     ! String in Datei schreiben
Close #99                           ! Datei schließen
Return
'
Procedure Ersetzer
  ' Null-String-Packer (Ergänzung zur Rplc-Routine)
  ' ----------------------------------------------------------
  Rplc(1,1,Buff$,"X0$+X0$","X02$",*Buff$)
```

```
    Rplc(1,1,Buff$,"X0$+X0$+X0$","X03$",*Buff$)
    Rplc(1,1,Buff$,"X02$+X0$","X03$",*Buff$)
    Rplc(1,1,Buff$,"X0$+X0$+X0$+X0$","X04$",*Buff$)
    Rplc(1,1,Buff$,"X03$+X0$","X04$",*Buff$)
    Rplc(1,1,Buff$,"X02$+X0$+X0$","X04$",*Buff$)
    Rplc(1,1,Buff$,"X02$+X02$","X04$",*Buff$)
    '
    ' Für einen Rplc-Aufruf in V3.0 ist ausschließlich das
    ' Pointer-Sternchen vor der hinteren Buff$-Übergabe
    ' (Rückgabe) zu entfernen.
  Return
  '
  Procedure Rplc(R.flg%,Pos%,M.str$,S.str$,R.str$,R.adr%)
    ' Für V3.0 ist folgender Kopf gültig:
    ' Procedure Rplc(Rflg%,Rpos%,Mstr$,Sstr$,Rstr$,Var Radr$)
    ' -------------------------------------------------------
    ' Setzt an Stelle des gefundenen Such-Strings den
    ' angegebenen Ersatz-String. Hier wird die Routine als
    ' Ergänzung zur Scode-Routine eingesetzt, sie ist
    ' jedoch als selbsständige Routine frei verwendbar.
    ' -------------------------------------------------------
    ' Rflg% = Flag, ob nur an jeweils erster gefundener
    '         Position ersetzt werden soll (Rflg%=0) oder
    '         alle gefundenen Positionen ersetzt
    '         werden sollen (Rflg%=1).
    ' Rpos% = Position innerhalb des zu durchsuchenden Strings,
    '         ab welcher gesucht werden soll.
    ' Mstr$ = Zu durchsuchender String
    ' Sstr$ = Zu suchender String
    ' Rstr$ = Einzusetzender String
    ' Radr% = Pointer auf eine Rückgabe-String-Variable, die
    '         nach Abschluß den neu gebildeten String enthält.
    '         In V3.0 ist dies eine VAR-String-Variable (Radr$),
    '         die den fertigen String aufnimmt.
    '
    Local Gf%,Ls$
    Repeat                            ! Check-Schleife
      Gf%=Instr(Rpos%,Mstr$,Sstr$)    ! Such-String suchen
      If Gf%                          ! Such-String gefunden
        Ls$=Left$(Mstr$,Gf%-1)        ! String-Anfang isolieren
        Mstr$=Ls$+Rstr$+Right$(Mstr$,Len(Mstr$)-Gf%-Len(Sstr$)+1)
        '                             ! Neuen Rest-String bilden
        Rpos%=Gf%+Len(Rstr$)+1        ! Neue Suchstartposition
      Endif
    Until Gf%=0 Or Rflg%=0            ! Fertig
    *Radr%=Mstr$                      ! String-Rückgabe in V2.xx
    ' Radr$=Mstr$                     ! String-Rückgabe in V3.0
  Return
```

READ { REA } — DATA-Werte auslesen

READ Var [,Var2,Var$,Var2$,...]

Der/Den angegebenen Variablen Var werden die jeweils gelesenen DATA-Einträge zugeordnet. Wird kein RESTORE Labelname verwendet, werden der Reihe nach vom Programmanfang aus soviele DATAs

eingelesen (falls vorhanden), wie READ-Anweisungen ausgeführt werden. Weiteres siehe unter DATA.

REM { R oder '} **Kommentar einfügen**

In V3.0 auch: { ! }

REM [Kommentar]

Möchten Sie Ihr Programm zur besseren Verständlichkeit mit Kommentar versehen, können Sie mit der Anweisung REM an beliebiger Programmstelle eine beliebige Kommentarzeile einfügen. Diese Kommentarzeilen bleiben vom Interpreter im Programmverlauf unberücksichtigt. Als Abkürzung können Sie für REM auch das Hochkomma (Apostroph) verwenden.

In der V3.0-Version kann auch ein Ausrufungszeichen (Kommentar-Marker innerhalb einer Befehlszeile) eingesetzt werden. Dieses wird - sofern es sich am Zeilenanfang befindet - in das REM-Zeichen ' umgewandelt.

Es ist empfehlenswert, diese Möglichkeiten zur Kommentierung ausgiebig zu nutzen. Bei größeren Programmen erleichtert es Ihnen das Auffinden von bestimmten Programmteilen. Außerdem machen kleine Bemerkungen die Logik Ihrer Programmführung und der Zusammenhänge auch für andere überschaubarer.

Soll allerdings das Programm, nachdem es umfassend kommentiert ist, auf die kürzestmögliche Länge wieder reduziert werden (z.B. zwecks kürzerer Ladezeiten, Speicherplatz-Ersparnis etc.), so ist ein sogenannter REM-Killer äußert hilfreich. Es ist nämlich eine äußerst strapazierende Geduldsleistung, aus einem großen Programm sämtliche Kommentare "zu Fuß" zu entfernen.

Nachfolgend finden Sie einen solchen REM-Killer, der bei seiner Arbeit alle Zeilen und Zeilenteile löscht, die entweder am Anfang mit dem REM-Zeichen ' oder REM beginnen oder am Zeilenende mit ! beginnen und keine An/Abführungszeichen mehr hinter sich haben. Mit einem nachfolgenden An/Abführungszeichen wäre es sehr wahrscheinlich ein Text-String, der nicht gelöscht werden darf. Da in DATA-Zeilen Strings auch ohne An/Abführungszeichen möglich sind, werden diese generell ausgeklammert.

Das Programm ist durch die vielen Sicherheitsabfragen etwas in die Länge gezogen, aber ich meine, daß sich dieser Mehraufwand sicher lohnt. Bei längeren Programmen kann die "Ent-REM-mung" schon einige Zeit in Anspruch nehmen, und es wäre ärgerlich, wenn die fertige Arbeit dann durch "Diskette voll" verlorenginge oder man aus Versehen ein Programm mit gleichem Namen beim Abspeichern überschreibt.

```
al$="Bitte zu 'ent-REM-mende|Quelldatei angeben!"
ALERT 1,al$,1,"OKAY",back%
FILESELECT "\*.LST","",quell$  ! Quelldatei-Auswahl
IF EXIST(quell$)               ! Datei existiert?
  OPEN "i",#1,quell$           ! Dann öffnen
  DIM puffer$(10000)           ! DIM Zeilen-Pufferfeld (max. 10000)
  '
  '     Die folgende WHILE..WEND-Schleife kann in V3.0 durch
  ' RECALL #1,puffer$(),10000,count%
  '     ersetzt werden.
  '
  WHILE NOT EOF(#1)            ! Noch kein Dateiende?
    LINE INPUT #1,puffer$(count%) ! Zeile lesen
    INC count%                 ! Zeilenzähler +1
  WEND                         ! Nächste Zeile
  '
  FOR cnt%=0 TO count%
    '
    ' In V3.0 reicht anstatt der nächsten drei Zeilen:
    '  ->    puffer$(cnt%)=TRIM$(puffer$(cnt%))
    '
    WHILE LEFT$(puffer$(cnt%))=" " ! Zeilenanfang = Space?
      puffer$(cnt%)=RIGHT$(puffer$(cnt%),LEN(puffer$(cnt%))-1)
      '                        ! Dann Space löschen
    WEND                       ! Nächstes Zeichen
    '
    IF LEFT$(puffer$(cnt%))<>"'" AND UPPER$...
            ...(LEFT$(puffer$(cnt%),3))<>"REM"  ! Keine REM-Zeile?
      IF UPPER$(LEFT$(puffer$(cnt%),4))<>"DATA" ! Keine DATA-Zeile?
        CLR p2%
        WHILE INSTR(p2%+1,puffer$(cnt%)," !") ! Ausrufungszeichen
          ' mit vorangestelltem Space enthalten? (muß mit Space
          ' sein, da sonst auch hintenstehende Binär-Variablen
          ' gefunden werden).
          p1%=INSTR(p2%+1,puffer$(cnt%)," !") ! Position feststellen
          p2%=INSTR(p1%+2,puffer$(cnt%),CHR$(34)) ! Anführungszeichen
          ' hinter '!'?  (-> dann wäre '!' Teil eines Strings!)
          IF p2%=0             ! Kein " hinter ! gefunden?
            puffer$(cnt%)=LEFT$(puffer$(cnt%),p1%-1) ! Kommentar...
          ENDIF                                      ! ...abtrennen
          EXIT IF p2%=0        ! Exit, wenn kein " mehr gefunden
        WEND
        ADD summe%,LEN(puffer$(cnt%))+2 ! Längen (+ CR/LF) addieren
      ENDIF
    ELSE                       ! Es ist eine REM-Zeile!
      '
      ' Sollen nur die !-Kommentare gelöscht werden, aber nicht
      ' die REM-Zeilen, so ist dieser ELSE-Block zu löschen.
```

```
    '
    puffer$(cnt%)=""           ! Für V2.xx
    '
    '  DELETE puffer$(cnt%)    ! Für V3.0
    '  DEC count%              ! Für V3.0
    '  DEC cnt%                ! Für V3.0
    '
  ENDIF
NEXT cnt%                      ! Nächste Zeile
CLOSE                          ! Datei schließen
WHILE (summe%+(2*count%)+1000)>DFREE(0) AND back%<>2
  '                            ! Neue Länge > Diskplatz?
  al$="Disk-Platz reicht|nicht aus!|Disk wechseln!"
  ALERT 3,al$,1,"OKAY|ABBRUCH",back%
WEND                           ! Noch einmal versuchen
IF back%<>2
  al$="Bitte neuen Namen für|'ent-REM-mte' Datei angeben !"
  ALERT 1,al$,1,"OKAY",back%
  nochmal:                     ! Wiederholungs-Label
  FILESELECT "\*.LST","",ziel$ ! Zieldatei wählen
  IF ziel$>""                  ! Nicht ABBRUCH geklickt?
    IF ziel$<>".LST"           ! Gültiger Name?
      IF ziel$=quell$ OR EXIST(ziel$)
        IF ziel$=quell$        ! Zielname = Quellname?
          al$="'Quellname' = 'Zielname' !|"
          al$=al$+"Es wird keine 'BAK'-Datei|angelegt!"
        ENDIF
        IF EXIST(ziel$)        ! Zielname existiert schon?
          al$="'Datei mit diesem Namen|"
          al$=al$+"existiert bereits!"
        ENDIF
        ALERT 1,al$,1,"OKAY|ZURÜCK",back%
        IF back%=2             ! Neuen Zielnamen angeben?
          GOTO nochmal         ! Dann zurück
        ENDIF
      ENDIF
      OPEN "o",#1,ziel$        ! Zieldatei öffnen
      '
      '     Die folgende FOR..NEXT-Schleife ist in V3.0 durch
      ' STORE #1,puffer$(),count%
      '     zu ersetzen.
      '
      FOR i%=0 TO count%-1     ! Alle Pufferzeilen
        IF puffer$(i%)>""
          PRINT #1,puffer$(i%) ! In Zieldatei schreiben
        ENDIF
      NEXT i%                  ! Nächste Zeile
      '
      CLOSE                    ! Zieldatei schließen
    ELSE                       ! Ungültiger Dateiname !
      ALERT 1,"Ungültige Dateibezeichnung!",1,"Nochmal",back%
      GOTO nochmal             ! Noch einmal von vorn
    ENDIF
  ELSE                         ! ABBRUCH geklickt !
    ALERT 2,"Programm-Abbruch?",1,"OKAY|NEIN",back%
    IF back%=2                 ! Doch noch überlegt?
      GOTO nochmal             ! Dann von vorn
    ENDIF
  ENDIF
```

```
    ENDIF
  ELSE                           ! Quelldatei nicht gefunden !
    IF quell$<>""                ! ABBRUCH geklickt?
      ALERT 3,"Datei nicht gefunden !",1,"Return",back%
    ENDIF
  ENDIF
```

RESTORE { RES } DATA-Zeiger setzen

RESTORE [Labelname]

RESTORE ohne Angabe einer Marke bewirkt, daß der DATA-Zeiger grundsätzlich auf den ersten aller im Programm enthaltenen DATA-Einträge gerichtet wird. Durch RESTORE Labelname kann der DATA-Zeiger auf das erste DATA gerichtet werden, das auf das angegebene Label folgt.

Ab dort werden dann die vorhandenen DATAs eingelesen, bis ein neuer RESTORE-Befehl eingesetzt wird oder das letzte aller DATAs gelesen wurde. Weitere Informationen hierzu finden Sie unter DATA.

6.4 Variablendeklarationen

Version 3.0

DEFBIT { DEFBI } Boole-Variable(n) deklarieren

DEFBIT Defstring$

DEFxxx-Befehle dienen der globalen Deklaration von Variablentypen. Es können verschiedene Definitions-Strings verwendet werden, die alle Variablen mit der darin angegebenen Namensspezifikation dem entsprechenden Variablentyp zuordnen.

Diese Deklaration wird üblicherweise zu Programmbeginn ausgeführt, da sonst eine Unterscheidung zwischen deklarierten und frei definierten Variablen erschwert wird. Wird jedoch innerhalb des Programms eine neue Deklaration ausgeführt, ist die vorhergehende damit ungültig. Nach einer Deklaration ist es nicht mehr nötig, die dadurch betroffenen Variablennamen mit einem sogenannten Postfix (Erkennungssymbol, z.B. % für 4-Byte-Integer, ! für Boole-Variablen oder $ für String-Variablen) zu versehen.

Es ist trotzdem jederzeit möglich, einzelne Variablen separat zu definieren, auch wenn sie dieselbe Namensspezifikation wie eine globale

Deklaration aufweisen. Dazu ist dem separaten Variablennamen das entsprechende Postfix hinzuzufügen. Damit wird die so unmißverständlich gekennzeichnete Variable von der Deklaration ausgeschlossen. Separat gekennzeichnete Variablen haben generell Vorrang vor den globalen Deklarationen.

Defstring$ ist ein String (Konstante, Variable oder Ausdruck), durch welchen die Namensspezifikation festgelegt wird. Bei den folgenden Definitionsbeispielen sind die verwendeten Deftsring$ beliebig austauschbar.

Definitionen:

DEFBIT "a"
Alle Variablen, deren Name als ersten Buchstaben ein a trägt, sind hiermit - sofern nicht separat definiert (s.o.) - als Boole-Variablen (Var!) deklariert.

DEFBYT "b,c,g-l"
Alle Variablen, deren Name als erstes Zeichen ein b, c, g, h, i, j, k oder l trägt, werden als 1-Byte-Integer (Var|) deklariert.

DEFWRD "word"
Alle Variablen, deren Name mit den Buchstaben word beginnt, werden als 2-Byte-Integer (Var&) deklariert.

DEFINT "i1,i2,i3"
Alle Variablen, deren Name mit den Buchstaben i1, i2 oder i3 beginnt, werden als 4-Byte-Integer (Var%) deklariert.

DEFFLT "a-c,x-z"
Alle Variablen, deren Name als ersten Buchstaben ein a, b, c, x, y oder z trägt, werden als 8-Byte-Fließkommavariablen (Var#) deklariert.

DEFSTR "d-f"
Alle Variablen, deren Name als ersten Buchstaben ein d, e oder f trägt, werden als String-Variablen (Var$) deklariert.

Beispiel:

```
Defint "i"          ! Alle mit I beginnenden Variablen
'                   ! gelten ab jetzt als 4-Byte-Integer.
I_string$="XYZ"     ! Beliebige Zuweisung zu einer mit
'                   ! I beginnenden String-Variable.
Print I_string$     ! Es wird XYZ ausgegeben, da trotz
'                   ! der vorangegangenen Definition die
'                   ! direkt angegebene String-Spezifikation
'                   ! $ gültig bleibt.
I_4byte=1548892.88  ! Beliebige Zuweisung zu einer mit I
'                   ! beginnenden Variablen ohne Postfix.
Print I_4byte       ! Es wird 1548892 ausgegeben. Übergebene
'                   ! Nachkommastellen werden integriert. Die
'                   ! 4-Byte-Definition kommt zur Geltung.
```

Version 3.0

DEFBYT { DEFB } 1-Byte-Integervariablen deklarieren

DEFBYT Defstring$

Siehe Erläuterungen zu DEFBIT.

Version 3.0

DEFFLT { DEFFL } 8-Byte-Fließkommavariablen deklarieren

DEFFLT Defstring$

Statt DEFFLT kann im Editor auch DEFSNG oder DEFDBL verwendet werden. Diese Angaben werden vom Editor bei der Syntaxkontrolle in DEFFLT umgewandelt. Sonst siehe DEFBIT.

Version 3.0

DEFINT { DEFI } 4-Byte-Integervariablen deklarieren

DEFINT Defstring$

Siehe Erläuterungen zu DEFBIT.

Version 3.0

DEFSTR { DEFS } Zeichenkettenvariable(n) deklarieren

DEFSTR Defstring$

Siehe Erläuterungen zu DEFBIT.

Version 3.0

DEFWRD { DEFW } 2-Byte-Integervariablen deklarieren

DEFWRD Defstring$

Siehe Erläuterungen zu DEFBIT.

6.5 Unterprogramme

DEFFN Funktion definieren

DEFFN Funkt.name [(Var-Liste)] = Funkt.expr

DEFFN ist eine sehr komfortable Möglichkeit, numerische oder alphanumerische Funktionen kompakt zu definieren. "Funkt.name" steht für einen beliebigen Namen, mit welchem die Funktion durch FN (bzw. @) angesprochen werden kann. In der Wahl des Funktionsnamens werden Ihnen keinerlei Einschränkungen auferlegt. Sie können außerdem auch BASIC-Befehlsnamen oder Namen schon existierender Variablen verwenden. Dies ist möglich, da der Funktionsname untrennbar mit der unter ihm definierten Funktion in Bezug steht. Die Verwendung von Feld-Variablennamen ist allerdings nicht zulässig. Das erste Zeichen kann, anders als bei Variablen, auch eine Ziffer sein.

Dem Namen kann optional (wahlfrei) eine Liste von Variablennamen in Klammern angefügt werden (Var-Liste), deren Inhalte dann innerhalb der Funktion verarbeitet werden können. Die einzelnen Variablennamen sind dabei durch Komma voneinander zu trennen. Diese Parameterliste kann ohne weiteres Variablen unterschiedlicher Typen (Real, Integer, String etc.) beinhalten. Innerhalb der Funktion sind dann allerdings nur Operationen erlaubt, die dem verwendeten Funktionstypen entsprechen. D.h., wenn die Funktion als String-Typ deklariert wurde (z.B. DEFFN Funktion$...), können nur String-Operationen ausgeführt werden. Bei numerischen Funktionstypen können folglich nur numerische Operationen eingesetzt werden.

Haben Sie eine Parameterliste vorgesehen, müssen beim Funktionsaufruf dementsprechend viele und dem jeweiligen Variablentyp entsprechende Werte, Strings, andere Variablen oder Ausdrücke übergeben werden. Entsprechen die übergebenen Parametertypen nicht der Liste oder werden zuviel, bzw. zuwenig Parameter übergeben, erscheint eine Fehlermeldung.

Die angegebenen Listen-Variablen müssen jedoch nicht zwingend innerhalb der Funktion verwendet werden (Dummy). Andererseits können globale Variablen in der Funktion verwendet werden, die nicht in der Parameterliste stehen. Bei Aufruf der Funktion werden dann die aktuellen Inhalte der darin verwendeten Variablen angenommen und weiterverarbeitet. Existiert eine globale Variable mit gleichem Namen wie der einer Parameter-Aufnahmevariable, so kommt deren Inhalt nicht zur Geltung, da die Listen-Variable innerhalb der Prozedur als lokal verarbeitet wird. D.h., sie hat dann ausschließlich den Wert, der ihr bei Funktionsaufruf über die Parameterliste zugewiesen wurde und nach Rückkehr wieder den vorherigen globalen Inhalt.

Die Länge einer Funktion wird durch die maximale Eingabezeilenlänge (256 Zeichen) beschränkt. Reicht dieser Platz nicht aus, um eine Funktion komplett zu definieren, können aus einer Funktion heraus andere Funktionen aufgerufen werden. Man kann also Funktionen beliebig miteinander verketten. Eine Funktion kann demnach ohne weiteres ausschließlich aus Funktionsaufrufen und deren Verknüpfungen bestehen.

Funktionen können an jeder beliebigen Stelle des Programms definiert sein. Da bei Programmstart alle DEFFNs initialisiert werden, kann dies also auch am Programmende geschehen, selbst wenn der Funktionsaufruf FN (bzw. @) vor der Funktion auftritt. Zu Beginn des Programmlaufs durchsucht der Interpreter den Programmtext nach evtl. vorhandenen DEFFNs, und deren Inhalt ist ihm daher vor Ausführung der ersten Zeile bereits bekannt.

Vorsicht: Endlosschleifen, worin sich zwei Funktionen gegenseitig aufrufen, können auch durch die Break-Funktion <Control/Shift/Alternate> nicht mehr unterbrochen werden. Rekursive Aufrufe haben denselben Effekt.

Beispiele siehe FN, INKEY$ und PRINT.

FN {@} Funktion aufrufen

FN Funktionsname [(Parameterliste)]

Mit FN bzw. dessen Kürzel @ können selbstdefinierte Funktionen aufgerufen werden (in V3.0 auch mehrzeilige FUNCTIONs). Sollen Parameter an die Funktion übergeben werden, wird dem Aufruf in Klammern eine Parameterliste nachgestellt, innerhalb derer die einzelnen Parameter ggfs. durch Kommas voneinander zu trennen sind. Wie bei DEFFN erwähnt, ist darauf zu achten, daß diese Angaben den in der DEFFN-Var-Liste aufgeführten Variablentypen entsprechen.

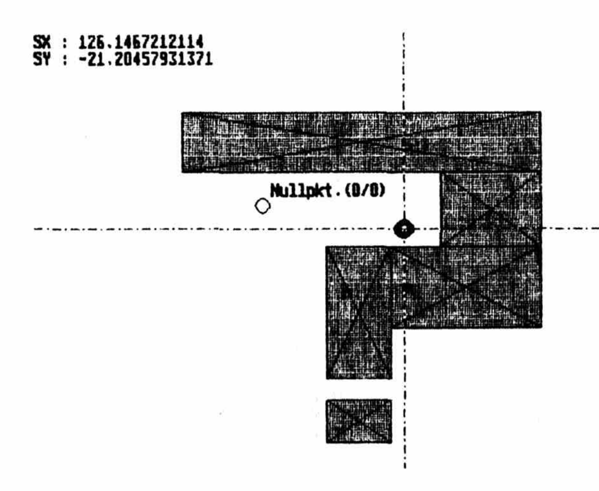
Das Ergebnis einer Funktion kann - wie bei jeder anderen BASIC-Funktion auch - über einen Ausgabe-Befehl (PRINT, WRITE, TEXT, OUT) ausgegeben, durch eine Zuweisung einer dem Funktionstyp entsprechenden Variablen übergeben und in entsprechende Ausdrücke oder Bedingungsabfragen integriert werden.

Beispiel (für Hires und Midres): Es soll der gemeinsame Schwerpunkt mehrerer Flächen ermittelt werden. Die einzelnen Flächen sind Rechtecke verschiedener Größe, die auf dem Bildschirm plaziert werden können. Die Anzahl der Flächen kann beliebig groß sein.

Zuerst erscheint ein kleiner Kreis. Dieser stellt den Nullpunkt dar, auf den sich anschließend die Koordinatenangaben beziehen. Plazieren Sie ihn an einer beliebigen Position und drücken Sie die Maustaste einmal kurz. Der Nullpunkt ist fixiert.

Zu Anfang einer jeden Rechteckdefinition erscheinen die zum Nullpunkt relativen Koordinaten, sowie die Nummer der aktuellen Fläche. Drücken Sie die **linke** Maustaste, wird die aktuelle Mausposition als Ausgangspunkt für die Rechteckbestimmung angenommen. Bestimmen Sie nun **mit** gedrückter Maustaste die Größe der Fläche. An Mauszeigerposition werden nun die Breite und die Höhe der Fläche angezeigt. Wenn das Rechteck Ihren Vorstellungen entspricht, lassen Sie die Maustaste los und die Fläche wird gezeichnet.

Mit einem Klick auf die **rechte** Maustaste beenden Sie die Eingabe und der gemeinsame Schwerpunkt der bisher eingegebenen Flächen wird gezeichnet und seine zum Nullpunkt relativen X/Y-Koordinaten werden angezeigt. Das Programm wartet dann darauf, daß die Maustaste nochmals gedrückt wird. Nachdem das getan ist, können Sie den nächsten Flächensatz eingeben bzw. das Programm beenden.




```
Do                                   ! Großer Schleifenstart
  Cls                                ! Screen klar
  Clear                              ! Variablen löschen
  Yt%=Min(2,3-Xbios(4))              ! Y-Teiler für Midres
  Deffill ,2,2                       ! DEFFILL hellgrau
  Graphmode 3                        ! XOR-Modus
  Defline 1,3                        ! Dickere Linie
  Repeat                             ! Schleife für Nullpunkt
    Mouse X0,Y0,K                    ! Maus-Status
    Circle X0,Y0,7                   ! Kreis zeichnen
    Mstop(X0,Y0,0)                   ! Auf Mausbewegung warten
    Circle X0,Y0,7                   ! Kreis löschen
  Until Mousek                       ! Exit, wenn Maustaste gedrückt
  Graphmode 2                        ! Transparentmodus
  Defline 1,1                        ! Dünne Linie
  Circle X0,Y0,7                     ! Nullpunkt zeichnen
  Text X0+8,Y0-8,"Nullpkt.(0/0)"     ! und beschriften
  Pause 20                           ! Kleine Klickpause
  Do                                 ! Box-Eingabeschleife
    Graphmode 3                      ! XOR-Modus
    Inc I                            ! Box-Zähler +1
    Defline 4                        ! Fadenkreuz-Linienmuster
    Repeat                           ! Linke obere Ecke setzen
      Mouse X,Y,K                    ! Maus-Status
      Text X+1,Y-1,Str$(X-X0)+"/"+Str$(Y0-Y)+" ("+Str$(I)+")"
      Line 0,Y,639,Y                 ! Fadenkreuz zeichnen
      Line X,0,X,399/Yt%             !      "         "
      Mstop(X,Y,0)                   ! Auf Mausbewegung warten
      Text X+1,Y-1,Str$(X-X0)+"/"+Str$(Y0-Y)+" ("+Str$(I)+")"
      Line 0,Y,639,Y                 ! Fadenkreuz löschen
      Line X,0,X,399/Yt%             !      "         "
    Until Mousek                     ! Exit, wenn Maustaste gedrückt
    Repeat                           ! Box aufziehen
      Mouse Xx,Yy,Kk                 ! Maus-Status
      Box X,Y,Xx,Yy                  ! -----------.
      Defline 4                                   !
```

```
        Line 0,Yy,639,Yy                       !
        Line Xx,0,Xx,399/Yt%                   !
        Text Xx,Yy,Str$(Abs(Xx-X))+"/"+Str$(Abs(Yy-Y))
        Mstop(Xx,Yy,-1)                        !
        Line 0,Yy,639,Yy                       !- Box zeichnen
        Line Xx,0,Xx,399/Yt%                   !
        Text Xx,Yy,Str$(Abs(Xx-X))+"/"+Str$(Abs(Yy-Y))
        Defline 1                              !
        Box X,Y,Xx,Yy               !------------'
      Until Kk<>1                   ! Mausknopf <> 1
      Exit if Kk=2                  ! Rechte Maustaste gedrückt?
      Breite=Abs(Xx-X)              ! Boxbreite berechnen
      Hoehe=Abs(Yy-Y)               ! Boxhöhe berechnen
      X.diff=Min(X,Xx)-X0           ! Nullpunkt-X-Abstand
      Y.diff=Y0-Max(Y,Yy)           ! Nullpunkt-Y-Abstand
      Graphmode 2                   ! Transparentmodus
      Pbox @X.lo,@Y.lo,@X.ru,@Y.ru ! Box zeichnen
      Line @X.lo,@Y.lo,@X.ru,@Y.ru ! Diagonalkreuz
      Line @X.lo,@Y.ru,@X.ru,@Y.lo !      "
      F.laeche=@Flaeche             ! Fläche berechnen
      Add Areas,F.laeche            ! Flächen addieren
      X.halbe=@Halbe(Breite)+X.diff ! Schwerpunkt-Abstand...
      Y.halbe=@Halbe(Hoehe)+Y.diff ! ... zum Nullpunkt
      Add An.xn,@Multi(X.halbe)     ! X-"Drehmomente" addieren
      Add An.yn,@Multi(Y.halbe)     ! Y-"Drehmomente" addieren
    Loop
    Graphmode 1                     ! Replace-Modus
    Sx=An.xn/Areas                  ! Gesamtschwerpunkt-...
    Sy=An.yn/Areas                  ! ...Abstände berechnen
    Defline 4                       ! Achsen-Linienmuster
    Line 0,Y0-Sy,640,Y0-Sy          ! Schwerpunkt-Achsen...
    Line X0+Sx,0,X0+Sx,399          ! ...zeichnen
    Defline 1,3                     ! Dickere Linie
    Circle X0+Sx,Y0-Sy,7            ! für Schwerpunkt-Kreis
    Print At(1,1);"SX : ";Sx        ! X-Abstand ausgeben
    Print At(1,2);"SY : ";Sy        ! Y-Abstand ausgeben
    Repeat                          ! Auf Mausklick warten
      If Mousek                     ! Maustaste gedrückt?
        Al$="Nächste Eingabe?       "
        Alert 2,Al$,1,"JA|NEIN|ENDE",Antwort
      Endif
      If Antwort=3                  ! Abbruch?
        Edit                        ! Dann Programmende
      Endif
    Until Antwort=1                 ! Weiter?
  Loop
  Deffn Flaeche=Hoehe*Breite
  Deffn Halbe(Seite)=Seite/2
  Deffn Multi(H.seite)=H.seite*F.laeche
  Deffn X.lo=X0+X.diff
  Deffn Y.lo=Y0-Hoehe-Y.diff
  Deffn X.ru=X0+X.diff+Breite
  Deffn Y.ru=Y0-Y.diff
```

Das obige Programm verwendet die folgende Prozedur Mstop, um das leidige Flimmern bei bewegten Grafikobjekten zu unterdrücken. Sie stoppt dazu einfach die Programmausführung, solange die Maus nicht

von der vorgegebenen Position wegbewegt wird oder die angegebene Maustaste entweder gedrückt (M.key% = 1 bis 3) oder eine andere als die angegebene Maustaste gedrückt wurde (M.key% = -3 bis 0). Das eingebaute ON MENU sorgt derweil dafür, daß ggfs. auf ON MENU GOSUB-Verzweigungen reagiert wird. So banal die Prozedur wirkt, so verblüffend ist ihre Wirkung. Flimmerfreie, bewegte Grafik ist ein sehr gefragtes Thema. Es gibt über die XBIOS(5)-Funktion bzw. in V3.0 auch über RC_COPY noch weitere Möglichkeiten, sie zu verwirklichen. Schauen Sie sich dazu bitte Kapitel 11.2. "Flimmerfreie Grafik" an.

```
Procedure Mstop(M.xex%,M.yex%,M.key%)
  ' M.xex% = Zu überwachende X-Koordinate
  ' M.yex% = Zu überwachende Y-Koordinate
  ' M.key% = Zu überwachender Maustasten-Status
  '         1 bis 3  Abbruch, wenn die angegebene
  '                  Maustaste gedrückt wurde
  '        -3 bis 0  Abbruch, wenn die angegebene
  '                  Maustaste nicht gedrückt wird
  If M.key%=<0
    Repeat
      On menu
    Until Mousex<>M.xex% Or Mousey<>M.yex% Or Mousek<>Abs(M.key%)
  Else
    Repeat
      On menu
    Until Mousex<>M.xex% Or Mousey<>M.yex% Or Mousek=M.key%
  Endif
Return
```

Version 3.0

FUNCTION..RETURN..ENDFUNC

{ FU..RET..ENDF } **Funktion**

FUNCTION Name [(Var1,Var2%,Var3$,...[,VAR Var,...])]
... auszuführende Programmteile
RETURN Back
ENDFUNC

FUNCTION kennzeichnet den Anfang einer selbstdefinierten, mehrzeiligen Funktion. Es kann eine optionale Liste von lokalen Variablen (wie bei DEFFN und PROCEDURE) angegeben werden, welche ggfs. die durch FN (bzw. @) übergebenen Daten aufnehmen. Dabei ist darauf zu achten, daß die Variablen den übergebenen Parametern entsprechen.

Es ist auch möglich, durch VAR innerhalb dieser Liste Variablen zu definieren, an die dann durch FN eine globale Variable direkt überge-

ben werden kann (siehe VAR, PROCEDURE). Außerdem ist - wie bei PROCEDURE - die Übergabe von indirekten Pointer-Parametern möglich (siehe auch GOSUB).

"Name" ist ein beliebiger Name, der die Funktion benennt. Innerhalb der FUNCTION kann beliebig viel Programmtext angeordnet werden. Im Gegensatz zu DEFFN-Funktionen ist es möglich, FUNCTION-Funktionen sich selbst aufrufen zu lassen (Rekursion).

Grundsätzlich ist eine FUNCTION mit einer PROCEDURE vergleichbar. Ein Unterschied ist, daß ENDFUNC (wie RETURN bei PROCEDURE) zwar den strukturellen Abschluß einer FUNCTION-Funktion bildet, jedoch im Gegensatz zu PROCEDURE...RETURN diese Endmarkierung nicht als Rücksprunganweisung interpretiert wird. Aus FUNCTION-Funktionen ist nur ein Rücksprung durch RETURN Back möglich, was nicht mit dem Prozedurende RETURN verwechseln werden darf.

Trifft das Programm innerhalb einer FUNCTION auf die Rücksprunganweisung RETURN Back, wird der hinter RETURN stehende Wert, Ausdruck oder Variableninhalt Back als Funktionsergebnis an das Programm zurückgegeben und zu dem dem Aufruf folgenden Befehl gesprungen. Es können beliebig viele RETURN Back-Anweisungen innerhalb einer FUNCTION angegeben werden (z.B. innerhalb von IF..ELSEIF..ELSE..ENDIF-Abfragen). Wie auch FN-Aufrufe einzeiliger Funktionen (DEFFN) kann ein Aufruf mehrzeiliger FUNCTIONs ebenfalls beliebig in Ausdrücke und Bedingungen eingebunden bzw. als Variablenzuweisung verwendet werden.

Soll die Funktion alphanumerische (String-)Ergebnisse liefern, ist dem Funktionsnamen ein $ anzuhängen (z.B. FUNCTION Name$).

Beispiel:

```
A$="1 ** UPPER/LOWER-Test-String ** 1"
Print "NORMAL: ";A$
Print "UPPER$: ";Upper$(A$)
Print "LOWER$: ";@Lower$(A$)
'
Function Lower$(L.str$)
  ' Untersucht einen vorgegebenen String auf Großbuchstaben.
  ' Gefundene Großbuchstaben werden in die entsprechenden
  ' Kleinbuchstaben umgewandelt.
  ' L.str$ = umzuwandelnder String
  '
  Local L.k%
  For L.k%=1 To Len(L.str$)
    Select Mid$(L.str$,L.k%,1)
```

```
    Case "A" To "Z"
      Mid$(L.str$,L.k%,1)=Chr$(Asc(Mid$(L.str$,L.k%,1))+32)
      ' Mid$(L.str$,L.k%,1)=Chr$(Asc(Mid$(L.str$,L.k%,1)) Or 32)
    Case "Ä"
      Mid$(L.str$,L.k%,1)=Chr$(Asc(Mid$(L.str$,L.k%,1))-10)
    Case "Ö"
      Mid$(L.str$,L.k%,1)=Chr$(Asc(Mid$(L.str$,L.k%,1))-5)
    Case "Ü"
      Mid$(L.str$,L.k%,1)=Chr$(Asc(Mid$(L.str$,L.k%,1))-25)
    Endselect
  Next L.k%
  Return L.str$
Endfunc
```

An diesem kleinen Beispiel läßt sich leicht das fast unüberschaubare Einsatzgebiet mehrzeiliger Funktionen erkennen. Ihrer Phantasie sind da keine Grenzen gesetzt. Hätte Frank Ostrowski die RINSTR-Funktion nicht implementiert, wäre sie hiermit spielend zu verwirklichen. Soll hier gleichzeitig der Inhalt von A$ nach Funktionsende gegen den Lower-Case-String ausgetauscht werden, muß man dazu nur den FUNCTION-Kopf folgendermaßen ändern:

```
Function Lower$(Var L.str$)
```

Übrigens soll die REM-Zeile Mid$(... im Beispiel einen kleinen Trick verdeutlichen. Die beiden MID$()-Zeilen unter "A" To "Z" lassen sich beliebig austauschen. Wissenswert ist, daß der einzige Unterschied zwischen Groß- und Kleinbuchstaben das Bit 5 (2^5=32) des ASCII-Wertes ist. Ist dieses Bit gesetzt, so handelt es sich um Kleinbuchstaben (ASCII: 97 - 122). Ist es gelöscht, so sind es folgerichtig Großbuchstaben (ASCII: 65 - 90).

Ähnliches gilt für normale Ziffernzeichen (ASCII: 48 - 57) und die entsprechenden Digital-Zeichen (ASCII: 16 - 25).

Beispiel:

```
Open "",#1,"VID:"
Print "Digitalziffern: ";@Digit$(Str$(1234567890));
Close
'
Function Digit$(Dig$)
  Local Z.k%
  For Z.k%=1 To Len(Dig$)
    Select Mid$(Dig$,Z.k%,1)
      Case "0" To "9"
      Mid$(Dig$,Z.k%,1)=Chr$(Asc(Mid$(Dig$,Z.k%,1)) Xor 32)
    Endselect
  Next Z.k%
  Return Dig$
Endfunc
```

In diesem Fall ist allerdings zu beachten, daß die Ausgabe kleinerer ASCII-Werte als 32 nur durch OUT oder VID: (siehe OPEN) möglich ist. Da hier der Aufruf-Parameter nicht als Variable, sondern direkt als Ausdruck angegeben wurde, ist eine VAR-Aufnahme im FUNCTION-Kopf nicht möglich.

Viele der hier im Buch gezeigten Utilities und Erweiterungen lassen sich auch in einer FUNCTION-Routine realisieren. Ich habe davon abgesehen, da ich davon ausgehe, daß bis zum Erscheinen des GFA-V3.0-Compilers doch die meisten von Ihnen (und viele auch noch danach) weiterhin überwiegend mit einer der V2.xx-Versionen arbeiten werden. Da die Routinen als PROCEDUREs auch ohne weiteres in V3.0 lauffähig sind, geht den V3.0-Programmierern dadurch nur sehr wenig an Qualität verloren.

GOSUB { GO oder @ } Verzweigung zu einer PROCEDURE

In V3.0: { G oder @ }

GOSUB Prozedur [(Parameterliste)]

Eine GOSUB-Verzweigung springt aus der GOSUB-Zeile direkt zum Kopf der in "Prozedur" bezeichneten PROCEDURE. Ist die Prozedur abgearbeitet, wird vom Prozedurende RETURN aus direkt zu der auf den entsprechenden GOSUB-Aufruf folgenden Programmzeile zurückgesprungen und dort das Programm fortgesetzt.

GFA-BASIC bietet Ihnen wahlweise auch die Möglichkeit, an eine Prozedur eine beliebig lange Parameterliste zu übergeben. Diese Parameter werden dort einer Liste von Variablen zugeordnet, die in Klammern im Prozedurkopf (z.B. PROCEDURE Test(Var,Var$,...)) angegeben sind. Die Anzahl der GOSUB-Parameter bzw. der PROCEDURE-Aufnahmevariablen ist nur durch die maximale Programmzeilenlänge von 256 Zeichen begrenzt.

Typen und Reihenfolge der Parameter in Parameterliste können beliebig gemischt werden, solange darauf geachtet wird, daß Anzahl, Typ und Listenplatz der Parameter mit Anzahl, Typ und Listenplatz der in PROCEDURE angegebenen Aufnahmevariablen übereinstimmen.

Durch den Sternchen-Pointer (*Var) können Variablen auch indirekt adressiert werden. Das bedeutet, daß ein in der Parameterliste angegebener Pointer die Adresse seiner Variablen an die Prozedur übergibt und nicht deren Inhalt. Wird nun innerhalb der PROCEDURE der

dafür vorgesehenen Aufnahmevariablen ebenfalls durch einen Pointer eine Variablenadresse zugewiesen, so finden Sie nach Abschluß in der übergebenen Variablen den durch den internen Pointer zugewiesenen Wert.

Beispiel:

```
@Reso(*Xt%,*Yt%)                ! Globale Adressen übergeben
Box 0,0,639/Xt%,399/Yt%         ! BOX in Bildschirmgröße
Procedure Reso(Xptr%,Yptr%)     ! Kopf mit lokalen Variablen
  ' Liefert die jeweils notwendigen Auflösungsteiler für X und Y
  ' Xptr% = Pointer auf eine Integervariable, die nach Abschluß
  '         den X-Teiler enthält.
  '         Hires/Midres = 1 ; Lowres = 2
  ' Yptr% = Pointer auf eine Integervariable, die nach Abschluß
  '         den Y-Teiler enthält.
  '         Hires = 1; Midres/Lowres = 2
  *Xptr%=2-Sgn(Xbios(4))        ! X-Teiler zurückgeben
  *Yptr%=Min(2,3-Xbios(4))      ! Y-Teiler zurückgeben
Return
```

Die im Prozedurkopf aufgeführten Variablen gelten innerhalb der Prozedur als lokale Variablen. D.h., daß die in ihnen enthaltenen Werte oder Strings nur innerhalb dieser Prozedur eingesetzt oder abgefragt werden können. Benennen Sie außerhalb dieser Prozedur globale Variablen mit dem gleichen Namen, so ist das BASIC in der Lage, diese von den internen, lokalen Variablen zu unterscheiden. Nach Rücksprung aus der Prozedur in das Hauptprogramm werden die globalen Inhalte wieder restauriert.

Zur Bildung des Prozedurnamens sind alle normalen alphanumerischen und numerischen Zeichen (A - Z und 0 - 9) sowie der Tiefstrich (Underscore _) und oder Punkt erlaubt. Anders als bei Variablennamen kann hier auch eine Ziffer als erstes Zeichen verwendet werden.

In Version V2.02 braucht nur noch der Prozedurname (evtl. mit Liste) angegeben zu werden. @ wird durch den Interpreter selbsttätig ergänzt. Aus proc(a%...) wird @Proc(A%...). In Version V3.0 ist die Angabe von GOSUB oder @ in den meisten Fällen überflüssig. Es wird vom Interpreter auch nicht mehr automatisch ergänzt. Sind Verwechslungsmöglichkeiten mit Befehlsnamen möglich (z.B. @RETURN, @RUN) ist die Angabe von @ allerdings zwingend notwendig.

Außerdem kann in V3.0 statt eines indirekten Pointer-Parameters (*Var) auch die Variable direkt übergeben werden. Dazu muß an entsprechender Stelle in der PROCEDURE-Aufnahmevariablenliste eine VAR-Variable gleichen Typs vorgesehen werden (siehe VAR).

Ein weiteres GOSUB-Beispiel finden Sie unter PROCEDURE.

GOTO { GOT } Unbedingter Sprung zu einem Label

GOTO Label

Dies ist in herkömmlichen BASIC-Dialekten einer der am häufigsten verwendeten Befehle. In GFA-BASIC hält sich dagegen seine Verwendung in Grenzen, da in den überwiegenden Fällen durch GOSUB und FN der erzielte Effekt wesentlich eleganter erreichbar ist.

Er bewirkt nichts weiter, als daß das Programm von der Zeile aus, in welcher es auf diesen Befehl trifft, zu der Programmzeile verzweigt, die mit dem angegebenen Label-Namen markiert ist. Dieses Label kann sich aus beliebigen Zeichen zusammensetzen. Genau wie bei Prozedurnamen ist es auch möglich, eine Ziffer als erstes Zeichen zu verwenden Der Label-Name muß mit einem nachgestellten Doppelpunkt abgeschlossen werden. Speziell in V2.02 und V3.0 ist hierauf besonders zu achten, da sonst das Label als Prozedur-Aufruf interpretiert wird.

Dasselbe Label kann übrigens auch als Sprungziel für RESTORE Label verwendet werden. Sprünge in oder aus FOR..NEXT-Schleifen oder PROCEDUREn sind nicht erlaubt. Wird dies versucht, so erscheint eine Fehlermeldung.

Beispiel (FOR..STEP..NEXT-Simulation für Hires/Midres):

```
Yt%=Min(2,3-Xbios(4))             ! Y-Auflösungsteiler
Clr I%                            ! 1. Schleifenzähler klar
I_stp%=40                         ! 1. Schrittweite
I_startlabel:                     ! 1. Start-Label
Clr J%                            ! 2. Schleifenzähler klar
J_stp%=40/Yt%                     ! 2. Schrittweite
J_startlabel:                     ! 2. Start-Label
Box 10+I%,10+J%,10+I%+20,10+J%+20 ! Grafik...
Box 12+I%,12+J%,12+I%+16,12+J%+16 ! ...ausgeben
Add J%,J_stp%                     ! 2.Zähler+2.Schrittweite
If J%<=399/Yt%                    ! Innerer Endwert erreicht?
  Goto J_startlabel               ! Nein, dann wieder von vorn
Endif
Add I%,I_stp%                     ! 1.Zähler+1.Schrittweite
If I%<=639                        ! Äußerer Endwert erreicht?
  Goto I_startlabel               ! Nein, dann wieder von vorn
Endif
```

LOCAL { LOC } — Lokale Variablen deklarieren

LOCAL Loc.var [,Lokale Variablenliste,...]

Es können innerhalb einer PROCEDURE - in V3.0 auch in einer FUNCTION - beliebige Variablen als lokal deklariert werden. Diese können dann ausschließlich in der Prozedur verwendet werden, in der sie deklariert wurden bzw. in den von dieser Prozedur aufgerufenen Unter-Prozeduren Für diese Unter-Prozeduren ist dann die lokale Variable der höheren Ebene gleichbedeutend mit globalen Variablen.

Wird im Hauptprogramm bzw. in Routinen höherer Ebene eine Variable mit gleichem Namen benannt, so ist der Interpreter in der Lage, diese von den lokalen Variablen der aktuellen Routine zu unterscheiden (siehe PROCEDURE).

Werden mehrere Variablen gleichzeitig deklariert, können sie unterschiedlichen Typs sein und sind dann durch Kommata voneinander zu trennen.

ON ... GOSUB — Bedingte Verzweigung zu Prozeduren

ON Wert GOSUB Proc1 [,Proc2,Proc3,...]
ON Wert Proc1 [,Proc2,Proc3,...] **(nur V3.0)**

Mit diesem Befehl lassen sich mehrere Prozeduren zu einer Liste zusammenfassen, die dann je nach angegebenem 'Wert' (numerischer Ausdruck, Variable oder Konstante) aufgerufen werden

Die Entscheidung darüber, zu welcher Prozedur verzweigt werden soll, wird nur in Einerschritten ab 1 aufwärts getroffen. Eine Verzweigung mit Wert = 0 ist nicht möglich. Die Verzweigung erfolgt also nach folgendem Schema:

```
Wert  >=  1  <  2  ==>  Proc1 aufrufen
Wert  >=  2  <  3  ==>  Proc2 aufrufen
Wert  >=  3  <  4  ==>  Proc3 aufrufen
usw.
```

Ist Wert größer als die Anzahl der angegebenen Prozeduren oder Wert gleich Null, so wird das Programm ohne Verzweigung in der auf ON GOSUB folgenden Programmzeile fortgesetzt.

Bei sämtlichen ON..GOSUB-Varianten des GFA-BASIC sind nur Prozeduren als Ziel erlaubt, die keine Parameterliste erwarten.

In V3.0 kann der Befehlsteil GOSUB weggelassen werden. Er wird vom Editor selbstständig eingefügt.

Beispiel:

```
On Menu(1)-20 Gosub Top,Clos,Full,Leer,Leer,Leer,Siz,Mov
Procedure Top
  ' Maßnahmen zum Aktualisieren eines Fensters
Return
Procedure Clos
  ' Maßnahmen zum Schließen eines Fensters
Return
Procedure Full
  ' Maßnahmen zum Dimensionieren eines Fensters auf
  ' größtmögliche Ausmaße.
Return
Procedure Siz
  ' Maßnahmen zur beliebigen Dimensionierung eines Fensters
Return
Procedure Mov
  ' Maßnahmen zum Bewegen eines Fensters
Return
Procedure Leer
  ' Dummy-Prozedur ohne Inhalt, die nur dann aufgerufen
  ' wird, sobald ein ON..GOSUB-Wert auftritt, der ohne
  ' Auswirkung bleiben soll. In diesem Fall würden bei
  ' den MENU(1)-Werten 23, 24 und 25 die slider- und
  ' pfeilbehandelnden Prozeduren aufgerufen werden. Zu
  ' Demo-Zwecken sollen diese Operationen hier jedoch
  ' unterdrückt werden.
Return
```

ON BREAK [CONT] [GOSUB] **Break-Funktion behandeln**

ON BREAK GOSUB Prozedur
ON BREAK ON BREAK CONT

Verzweigt im ersten Fall zu der angegebenen Prozedur, falls nach Ausführung des Befehls im Programmlauf die Break-Funktion <Control><Shift><Alternate> verwendet wird.

Im zweiten Fall wird die normale Break-Funktion aktiviert. D.h., daß Programm wird nach <Control><Shift><Alternate> beendet.

Die dritte Syntaxform bewirkt, daß das Programm bis zum nächsten ON BREAK oder ON BREAK GOSUB bzw. bis zum Programmende nicht mehr durch die Break-Funktion unterbrochen werden kann. Diese Möglichkeit ist ggfs. sehr vorsichtig zu handhaben, da bei endlos verzweigenden Strukturen ein Programmabbruch dann nur noch über

speziell vorgesehene Abbruch-Bedingungen oder durch die gefürchtete Reset-Taste möglich ist.

In V3.0 kann der Befehlsteil GOSUB weggelassen werden. Er wird vom Editor selbstständig eingefügt.

Beispiel:

```
On break gosub Break        ! Break-Funktion umleiten
Print At(1,1);"1. Stufe"
Print At(1,2);"2. Stufe durch <Control><Shift><Alternate>+<Esc>"
Do                          ! Endlos-Schleife
  If Ex%=1                  ! 2. Stufe schon erreicht?
    If Inkey$=Chr$(13)      ! <Return> gedrückt?
      Cls
      Print At(1,1);"3. Stufe"
      Print At(1,2);"Abbruch durch <Control><Shift><Alternate>"
      On break              ! Break-Funktion wieder einschalten
    Endif
  Endif
Loop
Procedure Break             ! Break-Routine (ist nur
  '                         ! durch Break-Tasten erreichbar)
  If Inkey$=Chr$(27)        ! Gleichzeitig <Esc> gedrückt?
    Ex%=1                   ! Flag für Stufe 3 setzen
    Cls
    Print At(1,1);"2. Stufe"
    Print At(1,2);"3. Stufe durch <Return>"
    On break cont           ! Break-Funktion ausschalten
  Endif
Return
```

PROCEDURE { PRO }...RETURN { RET } — Prozedur-Titel

In V3.0 auch:

SUB { SU }...ENDSUB { ENDSU }

Oder:

PROCEDURE { PRO }...ENDPROC { ENDP }

PROCEDURE Name [(Variablenliste)]
... auszuführende Programmteile
RETURN

Mit PROCEDURE werden in GFA-BASIC Unterprogramme bezeichnet. Diese können beliebigen Programmtext enthalten und werden durch die Rücksprunganweisung RETURN abgeschlossen. Mit RE-

TURN kehrt das Programm zu der aufrufenden Programmzeile zurück und fährt dort mit der darauffolgenden Programmzeile fort.

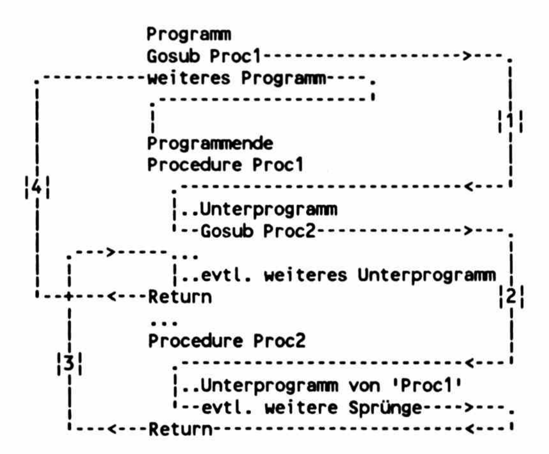
Es ist nicht sehr rationell, bestimmte Programmabläufe, die in immer derselben Form mehrfach im Hauptprogramm erforderlich werden, immer wieder neu zu formulieren. Dieses Verfahren ist also immer dann vorteilhaft, wenn allgemein gehaltene Unterprogramme von mehreren Programmstellen aus aufgerufen werden sollen.

Es kann optional eine Variablenliste lokaler Variablen (siehe LOCAL) angegeben werden, welche ggfs. die durch GOSUB übergebenen Daten aufnehmen. Die Variablen dieser Liste können unterschiedlichen Typs sein, wobei darauf zu achten ist, daß die Variablentypen den übergebenen Parametertypen entsprechen (bzw. umgekehrt).

Der Name der Prozedur kann aus beliebigen Zeichen (A - Z) und/oder Ziffern (0 - 9), dem Tiefstrich (_) und/oder dem Punkt (.) gebildet werden. Anders als bei Variablennamen kann hier auch das erste Zeichen aus einer Ziffer bestehen.

Es ist auch möglich, Prozeduren sich selbst aufrufen zu lassen (Rekursion, siehe Beispielprogramm unter FSNEXT()) bzw. aus einer Prozedur heraus beliebig weitere Prozeduren aufzurufen.

Struktur:



In Version V3.0 ist es auch möglich, durch VAR innerhalb der Variablenliste Variablen zu definieren, an die dann durch GOSUB eine globale Variable direkt übergeben werden kann (siehe VAR). Außer-

dem kann in Version V3.0 statt PROCEDURE auch SUB und statt RETURN auch ENDPROC oder ENDSUB verwendet werden. Diese Ausdrücke werden vom Interpreter in PROCEDURE bzw. RETURN umgewandelt.

Beispiele zu diesen beiden Struktur-Befehlen finden Sie in diesem Buch in Hülle und Fülle. Außerdem finden Sie weitere Informationen darüber unter FUNCTION, LOCAL und GOSUB.

Version 3.0

VAR — Direkte Variablen-Übergabe

PROCEDURE Name([Var1,...] VAR Varname1 [,Varname2$,...])
FUNCTION Name([Var1,...] VAR Varname1 [,Varname2$,...])

Als Alternative zu den in V2.xx ausgiebig genutzten indirekten Pointer-Variablen können innerhalb der Kopfzeilen von PROCEDUREs und FUNCTIONs in der Liste der lokalen Aufnahme-Variablen mittels VAR auch Variablen direkt an die Prozedur/Funktion übergeben werden. Die hinter VAR angegebenen Variablennamen stehen dann innerhalb der Prozedur/Funktion stellvertretend für die durch GOSUB oder FN übergebenen Variablen. Dabei ist zu beachten, daß die VAR-Variablen an letzter Stelle der Variablenliste placiert werden. Es werden dann **alle** Variablennamen, die hinter VAR stehen, als VAR-Variablen interpretiert.

Die VAR-Variable ist tatsächlich mit der globalen Variable identisch, sie trägt nur für die Dauer der Prozedur/Funktion den im Kopf angegeben Namen. Jede Veränderung der Aufnahmevariablen innerhalb der Prozedur wirkt sich also direkt auf die übergebene globale Variable aus.

Inhalte globaler Variablen, die denselben Namen wie die Aufnahmevariable tragen, bleiben erhalten. Der prozedur-/funktionsinterne Name hat also bis zum Rücksprung in das Hauptprogramm lokalen Charakter. Die gleichnamige globale Variable ist jedoch innerhalb der Routine nicht verfügbar. Nach Rücksprung zum Hauptprogramm (bzw. zur nächsthöheren Ebene) wird dann der globale Inhalt wieder restauriert.

Es ist dabei zu beachten, daß die beiden korrespondierenden Variablennamen dem gleichen Typ angehören. Ist das nicht der Fall oder wird eine Konstante oder ein Ausdruck als Parameter übergeben, erscheint die Fehlermeldung "Falsche VAR-Type".

Beipiele hierzu finden Sie in der Prozedur Pread unter der Beschreibung zu DATA, sowie in der Prozedur Astring unter der Beschreibung zu INSTR.

6.6 Assemler-/C-/PRG-Programmaufrufe

C:() Maschinenprogramm (C-compiliert) aufrufen

Var=C:Adressvar [(Parameterliste)]

Es ist in "Adressvar" die Adresse einer Maschinenroutine und - optional - eine Liste numerischer Parameter in Klammern zu übergeben. Die Übergabe erfolgt nach üblichen C-Konventionen auf dem Stack. Sollen 32-Bit-Parameter übergeben werden, ist dem jeweiligen Parameter das Kürzel 'L:' voranzustellen. Sonst gilt das Word-Format (16 Bit).

Beispiel:

```
VOID C:(L:Para1%,Para2%,W:Para3%,L:Para4%)
```

erzeugt folgenden Stack:

```
run:
move.l    (sp),return   ; = Rücksprungadresse
move.l   4(sp),para1    ; = 1. Parameter Para1% (Long)
move     8(sp),para2    ; = 2. Parameter Para2% (Word)
move    10(sp),para3    ; = 3. Parameter Para3% (Word)
... etc.
... Maschinenprogramm
rts                     ; Nur RTS verwenden !!
```

Sind keine Parameter zu übergeben, ist eine Leerklammer () zu verwenden. Nach Rückkehr kann der Inhalt von D0 (wie in C) entweder direkt ausgegeben (PRINT C:Var()), einer Variablen übergeben (A%=C:Var()) oder in Bedingungsabfragen eingebunden werden (IF C:Var()).

CALL { CA } — Maschinenprogramm (assembliert) aufrufen

In V3.0: { CAL }

CALL Adressvar [(Parameterliste)]

"Adressvar" enthält die Adresse der aufzurufenden Maschinenroutine (beachte DIM-Vermerk). "Parameterliste" enthält, durch Kommata getrennt, die evtl. zu übergebenden Parameter. Diese können beliebigen Typs sein, werden jedoch alle als 32-Bit-Werte interpretiert. Das Maschinenprogramm empfängt auf dem Stack der Reihe nach die Rücksprungadresse, einen 16-Bit-Wert, der die Parameteranzahl enthält, sowie eine 32-Bit-Adresse, ab welcher die Parameter im 32-Bit-Format aufeinanderfolgend vom BASIC abgelegt wurden. String-Parameter werden hier vom BASIC mit deren Anfangsadresse übergeben.

In den V2.xx-Versionen steckt hier ein kleiner BASIC-Fehler. Um die Parameter korrekt auf dem Stack zu erhalten, muß als letzter Parameter ein Null-Word angehängt werden. In V3.0 ist diese Maßnahme nicht mehr nötig, aber auch nicht schädlich. Bei V2.xx-V3.0-Anpassungen kann dieses Nullword ggfs. im Aufruf stehengelassen werden.

Beispiel:

```
' CALL Prog%(17,"Para",A%,B$,*Var%,0)  ! V2.xx-Aufruf
CALL Prog%(17,"Para",A%,B$,*Var%)      ! V3.0-Aufruf
```

erzeugt folgenden Stack:

```
run:
move.l    (sp),return   ; = Rücksprungadresse
move     4(sp),anzahl   ; = 5 Parameter
move.l   6(sp),adress   ; = Zeiger auf Parameterblock
move.l  adress,a1       ; = Zeiger nach a1
move   (a1),para1       ; = Parameter 1 in para1
move   4(a1),para2      ; = Parameter 2 in para2
... etc.
... Maschinenprogramm
rts                     ; Nur RTS verwenden!
```

Beispiel:

Die folgende Routine scrollt einen beliebigen Bildschirmausschnitt in horizontaler Richtung. Vor der ersten Benutzung der Scroll-Routine braucht nur einmal die Initial-Prozedur Psc aufgerufen werden. Diese liest den Maschinen-Code ein und liefert der BASIC-Scroll-Routine die nötigen Adressen in globalen Variablen.

Anschließend kann ohne weitere Vorbereitung jederzeit die Scroll-Routine mit variablen Parametern angesprungen werden. Im Farbmodus ist die Routine leider nicht einsetzbar, da der Auschnitt durchgehend wordweise gescrollt wird. Da im Farbbetrieb jedoch die Farbebenen immer "zwischen" den Bildschirm-Words liegen, kommen hier die Ausschnittkoordinaten und Farben durcheinander.

```
@Psc                         ! Routine initialisieren
Pbox 0,0,639,399
Deffill ,3,10
Pbox 15,120,624,165
Text 32,150,String$(31,"*")+"scroll-demo"+String$(30,"*")
@Scroll(16,120,624,165,-1) ! Aufruf
'
Procedure Scroll(Sc_xl%,Sc_yo%,Sc_xr%,Sc_yu%,Sc_ab%)
  ' ****** Nur für Hires ******
  ' Sc_xl%, Sc_yl%, Sc_xr%, Sc_yu%
  '        = Koordinaten des Ausschnitts. Die Angabe der
  '          X-Koordinaten ist nur mit durch 16 teilbaren
  '          Werten sinnvoll.
  ' Sc_ab% = Gibt an, um wieviel der Ausschnitt jeweils
  '          nach links verschoben werden soll.
  ' Vorsicht bei hohen Sc_ab%-Werten! Das Scrolling läßt sich
  ' auch durch <Control/Shift/Alternate> nicht unterbrechen
  Local Sc_ln%
  Sc_ln%=C:Sc_ad1%(Sc_xl%,Sc_yo%,Sc_xr%,Sc_yu%) ! Breite ermitteln
  If Sc_ab%>0                      ! Abgleich größer 0?
    Hidem                          ! Maus aus
    Call Sc_ad2%(Sc_ln%,Sc_ab%,0)! Scrolling
    Showm                          ! Maus an
  Endif
Return
Procedure Psc
  ' Scrollinit-Routine.
  Local Sc_i%,Sc_a$
  Restore Sc_roll.datas        ! DATA-Label setzen
  Void Fre(0)                  ! Garbage-Collection
  For Sc_i%=1 To 74            ! 74 DATAs
    Read Sc_a$                 ! lesen
    Sc_rt$=Sc_rt$+Mki$(Val("&H"+Sc_a$)) ! In Puffer schreiben
  Next Sc_i%
  Sc_ad1%=Varptr(Sc_rt$)       ! Startadresse des Init-Programms
  Sc_ad2%=Sc_ad1%+86           ! Startadresse des Scroll-Programms
  Sc_roll.datas:
  Data 3f3c,3,4e4e,548f,2040,43fa,4a,302f,8,e848,322f,4,e849
  Data 9041,e349,44c1,d1c1,3200,640,6fff,3340,20,e349,3341,26
  Data 302f,6,c0fc,50,d1c0,2348,14,302f,a,906f,6,5340,3340,1a
  Data 3001,e748,44c0,4e75,226f,6,2429,4,5382,2629,8,51cb,fffe
  Data 207c,7,8000,323c,a,3010,e350,7027,d1fc,0,50,e5e0,51c8
  Data fffc,d1fc,0,50,51c9,ffe6,51ca,ffd0,4e75
Return
```

EXEC { EXE } .PRG/.TOS/.TTP-Programm laden/starten

EXEC Modus,"Prg","Kom","Env" -> als Befehl
Var=EXEC(Modus,"Prg","Kom","Env") -> als Funktion

Modus:

```
0 = Programm laden u. starten
3 = Programm laden und Basepage liefern
4 = Programm starten
    (in Kom als MKL-String Basepage angeben Prg u. Env = "")
```

Aus einem GFA-BASIC-Programm heraus können Desktop-Programme geladen, reloziert und gestartet werden. Grundsätzlich tut der Befehl EXEC 0,... nichts anderes, als ein beliebiges Programm-File in den nächstmöglichen freien Speicher zu laden, es zu relozieren und gleichzeitig zu starten. Das ist ausschließlich mit kompletten PRG-, TOS- oder TTP-Programmen möglich, da diese eine eigene Speicherverwaltung beinhalten, bzw. der Programm-Header vom System für einen korrekten Programmstart verwendet werden kann.

Nach Ende des aufgerufenen Programms kehrt das System nicht zum Desktop, sondern zu aufrufenden BASIC-programm zurück, das in der Zwischenzeit unverändert im Speicher auf den Rücksprung gewartet hat. Das aufrufende Programm bleibt also resident.

Soll ein Programm nur geladen werden, muß dazu 'Var=EXEC(3,...)' verwendet werden, da nur so die Startadresse des Maschinen-Codes (Var+256) ermittelt werden kann. Der Code ist dann reloziert und kann durch C: oder CALL aufgerufen werden. Bei C-compilierten Programmen ist zu beachten, daß der geladenen Code 256 Bytes hinter der in Var gelieferten Adresse beginnt. Dies muß also auch die Einsprungsadresse des Programms sein. Evtl. vorgelinkte C-Header würden also zum Chaos führen.

Die erste Syntaxform (als Befehl) ist mit Modus = 0 nur dann sinnvoll, wenn kein Rückgabewert vom aufgerufenen Programm erwartet wird. Andernfalls kann mit der zweiten Form (als Funktion) und Modus = 0 (Var=EXEC(0,...) ein Wert vom aufgerufenen Programm empfangen werden. Dieser Wert muß vor Rückkehr vom aufgerufenenen Programm in D0 abgelegt worden sein. In Version V3.0 kann durch QUIT oder SYSTEM optional ein Rückgabewert angegeben werden. Dieser wird vom BASIC in D0 abgelegt und kann nach Rückkehr vom aufrufenden Programm ausgewertet werden.

Konnte EXEC aus irgendwelchen Gründen nicht ausgeführt werden (Speicher voll, Datei nicht gefunden) erhält man als Rückgabe die entsprechende Fehlernummer. Bei Modus = 3 ist nur die zweite Form sinnvoll, da die Basepage-Adresse (siehe Basepage) des geladenen Programms nur so in Erfahrung gebracht werden kann.

Vor Ausführung ist genügend Speicherplatz durch RESERVE freizumachen, da EXEC nur in noch nicht allozierte Bereiche lädt. Bei EXEC 0,... wird die Freigabe des vom Programm belegten Speichers vor Rücksprung automatisch durchgeführt und er kann dann durch RESERVE weiterverwaltet werden. Bei EXEC 3,... (als Befehl) ist die Speicher-Restauration dagegen völlig unmöglich, da keine Adresse bekannt ist, die durch MSHRINK() oder MFREE() freigegeben werden könnte. Die Funktion Var=EXEC(3,...) läßt dagegen eine Restauration anhand der in Var gelieferten Adresse zu. Wird das Programm nicht mehr benötigt, kann der davon belegte Speicher durch die GEMDOS-Funktion 73 (siehe MFREE()) oder in V3.0 durch MFREE(Var) wieder freigegeben werden.

Prg enthält den vollständigen Namen des zu ladenden Programms (inkl. Pfad, siehe unter 5.4 "Diskettenoperationen").

Kom enthält eine evtl. zu übergebende Kommandozeile, die (unter GFA-BASIC) in der zweiten Hälfte der Basepage des aufgerufenen Programms abgelegt wird (ab Basepage+128). Nach Standard-Konventionen enthält dabei das erste Byte der Kommandozeile deren Länge. Der Inhalt der Kommandozeile unterliegt keiner Konvention. Es können hier beliebige Informationen an das aufgerufene Programm übergeben werden (Dateinamen, Werte, Strings).

Env ist der Environment-String (siehe SHEL_ENVRN), der nach Standard-Konventionen durch je ein Null-Byte getrennte Informtionen enthält. Abgeschlossen wird dieser String durch ein doppeltes Null-Byte, was GFA-BASIC automatisch erledigt. Die Startadresse dieses Strings kann im geladenen Programm aus BASEPAGE+44 ausgelesen werden.

Soll Kom oder Env nicht übergeben werden, kann dafür ein Leer-String ("") verwendet werden. Nicht ausführbar sind z.B. speicherresidente Programme wie SID oder der SEKA-Assembler oder alle Programme, die in irgendeiner Form "Batches" verwenden.

Beispiele:

```
Reserve 1000                        ! Kleiner Restspeicher
Exec 0,"GFABASIC.PRG","Basprog.Bas","" ! BASIC starten
Reserve Xbios(2)-16384-Himem+Fre(0) ! Speicher restaurieren
' in V3.0 einfacher: RESERVE
...weiter im Programm
-----------------------------------------------------------------
Reserve 1000                        ! Kleiner Restspeicher
Back%=Exec(0,"Programm.Prg","","")  ! Programm starten und
'                                   ! Rückgabewert liefern
Reserve Xbios(2)-16384-Himem+Fre(0) ! Speicher restaurieren
' in V3.0 einfacher: RESERVE
Print Back%                         ! Rückgabewert ausgeben
...weiter im Programm
-----------------------------------------------------------------
Reserve 1000                        ! Kleiner Restspeicher
Adrs%=Exec(3,"Programm.Prg","","")  ! Programm laden und
'                                   ! Basepage-Adresse liefern
Rout%=Adrs%+256                     ! Code-Start berechnen
Back1%=Void C:Rout%(Var1,Var2)      ! Programm aufrufen
Back2%=Void C:Rout%(Var3,Var4)      ! Programm nochmal aufrufen
Print Back1%''Back2%                ! Rückgabewerte ausgeben
Void Gemdos(73,L:Adrs%)             ! MFREE-Reallocation
' In V3.0 einfacher: ~MFREE(Adrs%)
Reserve Xbios(2)-16384-Himem+Fre(0) ! Speicher restaurieren
' In V3.0 einfacher: RESERVE
...weiter im Programm
```

MONITOR { MON } — Maschinen-Programm aufrufen

In V3.0: { M }

MONITOR [(Parameter)]

Parameter enthält optional einen Übergabewert, der in D0 an die aufgerufene Routine geliefert wird. In den V2.xx-Versionen wird in A0 zusätzlich die Adresse einer 32-Byte-Tabelle (8 Longwords) übergeben, die auf die Speicherbereiche der verschiedenen Variablenarten zeigt (siehe TYPE). Diese Übergabe findet in V3.0 nicht mehr statt!

Der Befehl MONITOR erzeugt ganz banal eine Illegal Instruction-Exception. Dies ist ein Systemfehler, der immer dann auftritt, wenn in einem Maschinenprogramm versucht wird, einen Befehl auszuführen, den der ST nicht kennt. Das passiert meist dann, wenn der PC (ProgramCounter = CPU-Zeiger auf nächste auszuführende Adresse) aufgrund eines Programmfehlers in die Wüste zeigt. Im Normalfall würde das einen 4-Bomben-Absturz (mit folgendem Reset) nach sich ziehen, da das System standardmäßig in der gesamten Exception-Sprungtabelle (8 Longwords ab Adresse 8) immer denselben Zeiger stehen hat. Dieser zeigt auf eine Maschinenroutine, die die berühmten Bömbchen

"malt" und zwar je nach Exception 2 bis 9 Bomben "in a row". GFA-BASIC fängt diese Bombenwürfe ab, indem es in die Zeiger-Tabelle eigene Zeiger einträgt. Und zwar zeigen diese dann auf eine Routine, die eine entsprechende Alert-Box erzeugt und dann in das BASIC zurückspringt.

```
LPEEK(8)  =  Adresse der Busfehler-Routine
LPEEK(12) =  Adresse der Adreßfehler-Routine
LPEEK(16) =  Adresse der Illegal-Routine
LPEEK(20) =  Adresse der Nulldivision-Routine
LPEEK(24) =  Adresse der CHK-Routine
LPEEK(28) =  Adresse der TRAPV-Routine
LPEEK(32) =  Adresse der PRIVILEG-Routine
LPEEK(36) =  Adresse der TRACE-Routine
```

Die Idee hinter MONITOR ist nun die, daß erstens in GFA-BASIC eine Illegal-Instruction-Exception unter normalen Umständen unmöglich ist und zweitens, daß verschiedene Programme (z.B. SID oder TEMPLMON) ebenfalls die Bomben abfangen und die Exceptions dann dazu verwenden, in den Debugger, Disassembler, Maschinen-Monitor oder ähnliches zu springen. In diesem Fall hat das den Vorteil, daß man dem aufgetretenen Fehler auf Maschinenebene sofort nachspüren kann.

Aus all dem folgt, daß man natürlich auch vom BASIC aus diesen Zeiger auf eine eigene Maschinenroutine "verbiegen" kann. Voraussetzung für diese komfortable Art des Maschinenprogramm-Aufrufs ist:

a. daß der Illegal-Instruction-Vector (Exception-Vektor 4) auf die Adresse der eigenen Routine gerichtet wird.
b. daß die Routine nicht durch BASIC-Aktionen verschoben wird.
c. daß das HI-Byte des HI-Words (MSB) des Vektors vor BASIC-Start 0 ist (standardmäßig immer der Fall).
d. und daß die aufgerufene Routine - statt mit RTS - mit RTE zu enden hat.

Die erste Bedingung ist ganz einfach zu erfüllen, indem man durch SLPOKE 16,Adresse die Start-Adresse der eigenen Routine in den Exception-Vektor einträgt.

Das macht natürlich nur dann Sinn, wenn ab "Adresse" auch ein Maschinenprogramm zu finden ist. Damit sind wir bei der zweiten Bedingung, die schon nicht mehr ganz so einfach zu erfüllen ist. Die gebräuchlichste Variante ist, daß das Programm in einem String-Puffer untergebracht wird. Das hat den Nachteil, daß String- und Feldadres-

sen in GFA-BASIC durch DIM und ERASE verschoben werden können, wodurch die im Vektor eingetragene Adresse nicht mehr stimmt. Sicherer ist es, ein entsprechend großes Integerfeld einzurichten und dann die Routine ab VARPTR(Feld%(0)) zu installieren. In beiden Fällen muß der ursprüngliche Vektorinhalt zwischengespeichert werden, um ihn vor Verlassen des BASIC-Programms bzw. des Compilats wieder restaurieren zu können.

Die eleganteste Lösung besteht allerdings darin, vor BASIC-Start bzw. vor Start des Compilats einen ausreichenden Speicherbereich zu reservieren und durch KEEP PROCESS (GEMDOS-Funktion) resident zu halten. Wie das funktioniert, können Sie unter VST_LOADFONT() nachlesen. Nachdem dies geschehen ist, kann vom Desktop aus das BASIC oder Compilat gestartet und von dort aus beliebig über den reservierten Speicher verfügt werden. In diesem Fall kommt es auf die Konzeption der auszuführenden Maschinenroutine an, ob der Vektor restauriert werden muß oder nicht.

Um nicht nach Reservierung wieder zum Desktop zurückkehren zu müssen, kann auch aus dem Speicher-Reservierungsprogramm heraus durch die GEMDOS-Funktion $4B EXEC das BASIC oder Ihr Compilat gestartet werden.

Wie die Bedingungen c und d zu erfüllen sind, ergibt sich aus der Bedingungsstellung selbst.

Weitere Informationen zum Thema Exceptions bzw. MONITOR finden Sie z.B. in der Zeitschrift c't, Ausgabe 12/86 oder in "GFA-BASIC Tips & Tricks" von Data Becker.

Haben Sie keine der genannten Vorarbeiten ausgeführt und rufen ohne weitere Vorbereitung MONITOR auf, passiert nichts anderes, als daß die BASIC-Fehlermeldung "4 Bomben - Illegal Instruction" erzeugt wird und danach entweder zum Editor, Desktop oder in eine BASIC-Abfangroutine (siehe ON ERROR) verzweigt wird.

Die andere, oben schon angedeutete Möglichkeit, den MONITOR-Befehl zu nutzen, besteht darin, den GFA-BASIC-Interpreter bzw. ein GFA-compiliertes Programm von einem residenten Maschinen-Monitor aus zu starten. Die Voraussetzung ist hier, daß System-Fehler von diesem Monitor sinnvoll abgefangen werden. Sinnvoll heißt in diesem Fall, daß man nach Auftreten eines Errors dessen Ursache vom Monitor bzw. Debugger aus erforschen und nach erfolgter Reparatur das unterbrochene Programm fortsetzen kann.

Es gibt nun verschiedene Ausführungen solcher Debugger. Die bekanntesten und wohl auch verbreitetsten sind der Digital-Research-Debugger SID und der Tempelmann-Debugger TEMPLMON. Speziell zur Nutzung des MONITOR-Befehls müssen diese Monitore den erwähnten Illegal-Instruction-Vector entsprechend verbogen haben.

Wollen Sie den SID verwenden, starten Sie ihn wie gewohnt durch Anklicken des Desktop-Icons. Dabei ist darauf zu achten, daß der SID die Extension PRG trägt, da sonst GEM-Anwendungen nicht korrekt ausgeführt werden können (ggfs. ändern). Legen Sie nun die BASIC-Programm-Diskette ein.

```
e gfabasic.prg      -   Lädt den BASIC-Interpreter
g                   -   Startet den Interpreter
```

Beim ersten Eintritt in den Interpreter wird eine Fehlermeldung ausgegeben, die ignoriert werden kann. Sie ist ohne Auswirkung auf die Arbeit des Interpreters.

Dasselbe Verfahren ist mit compilierten Programmen möglich, wobei diese allerdings mit der Bomben-Abfang-Option compiliert worden sein müssen und zu Beginn eine ON ERROR GOSUB-Verzweigung vorsehen, da sonst aufgrund der anfänglichen Fehlermeldung das Programm umgehend beendet wäre.

Tritt nun während Ihrer Arbeit mit dem Interpreter oder während des Programmverlaufs ein Error auf, wird keine Error-Meldung ausgegeben, sondern das System springt zur SID-Eingabe-Ebene zurück. Von hier aus können Sie jetzt die Optionen des SID zur eigenen Fehlerbehandlung und -verfolgung nutzen (das D in SID steht für Debugger; debugging = engl.: entwanzen, Fehler ausmerzen). Die Eingabe von g setzt dann die Interpreterarbeit bzw. Ihr Programm fort.

Möchten Sie den Monitor direkt aufrufen, geschieht das eben durch den Befehl MONITOR. Sie gelangen dann durch die dadurch ausgelöste Illegal-Exception ebenfalls in die Eingabe-Ebene des SID. Bevor Sie jedoch nun zum Interpreter bzw. Programm zurückkehren können, muß vorher durch Eingabe von xpc im SID der Stand des Programmzählers (PC = Program-Counter) angefordert werden. Es wird nun der Inhalt des PC's auf dem Bildschirm ausgegeben und der SID wartet auf die Eingabe eines neuen PC-Inhalts. Zählen Sie zum ausgegebenen PC-Stand den Wert 2 hinzu und geben Sie den neuen Wert ein. Wenn Sie jetzt <RETURN> drücken, übernimmt der SID diesen Wert in den PC und kehrt zu seiner Kommando-Ebene zurück. Nachdem das voll-

bracht ist, können Sie wieder durch den Befehl g (Abk. für go) den Interpreter bzw. Ihr Programm aufrufen.

Einige Befehle des SID:

e Programmname
Programm laden (e = Abk. f. Execute).

g
Aktuelles Programm starten.

x
Anzeige sämtlicher CPU-Register.

x Register
Einzelregister zeigen und Änderung abwarten. Beispiele:

```
xs  =  Stack            (ST)
xss =  Superstack-Pointer(Ssp)
xus =  Userstack-Pointer (USP)
xa0 =  Adreßregister 0 (A0)
xd4 =  Datenregister 4  (D4)
       usw.
```

d Adresse
Speicher ab Adresse in Bytes zeigen.

dw Adresse
Speicher ab Adresse in Words zeigen.

dl Adresse
Speicher ab Adresse in Longs zeigen. Beispiel: dlb0ef2.

l Adresse
Ab Adresse disassemblieren.

l
Nächsten Block disassemblieren.

s Adresse
Byte ab Adresse zeigen und Änderung abwarten.

sw Adresse
Word ab Adresse zeigen und Änderung abwarten.

sl Adresse
Longword ab Adresse zeigen und Änderung abwarten.

Wird bei den Befehlen s, sw, sl <Return> gedrückt, ohne den Wert verändert zu haben, bleibt der alte Adresseninhalt bestehen und die nächste Adresse wird im gewünschten Format angezeigt. Zur Kommando-Ebene gelangt man wieder, indem man <Tab> <Return> drückt.

Eine nähere Beschreibung des TEMPLMON-Monitors (Public-Domain) entfällt hier, da dieser intern über eine ausführliche Bedienungsanleitung verfügt (<?> <Return> drücken). Dieser Monitor ist sehr empfehlenswert und kann gegen eine geringe Kostenbeteiligung (evtl. vorher erkundigen) mitsamt ausführlicher Bedienungsanleitung inkl. Tips und Tricks von Thomas Tempelmann, E.L.Kirchner Strasse 25, 2900 Oldenburg bezogen werden. Er wird im Auto-Ordner gestartet und ist dann jederzeit durch die Tastenkombination <Alternate>-<Help> verfügbar.

Version 3.0

RCALL { RC } Masch.-Programmaufruf m. Registerzugriff

RCALL Adresse,Feld%()

Ruft ein ab "Adresse" liegendes Maschinenprogramm auf. Dabei kann der Name eines 32-Bit-Integer-Arrays (mind. 16 Elemente) übergeben werden, in dem vor Aufruf beliebige Longwords abgelegt werden können. Die Inhalte der ersten 15 Elemente dieses Feldes (OPTION BASE beachten) werden vor Ausführung der Routine folgendermaßen in die CPU-Register kopiert (bei OPTION BASE 0):

```
Feld%(0) bis Feld%(7)  --> d0 bis d7
Feld%(8) bis Feld%(14) --> a0 bis a6
```

Nach Abschluß der Routine (mit RTS) werden die Registerinhalte in der gleichen Reihenfolge wieder in das Feld zurückgeschrieben. Zusätzlich erhält man in Feld%(15) (bei OPTION BASE 0, sonst in Feld%(16)) den aktuellen Userstack-Pointer (SP = a7 - nur Rückgabe!).

Beispiel (BMOVE-Simulation): Dieses Beispiel dient nur als Demonstration und erfüllt keine höheren Ansprüche. Bei Angabe der Adressen wird nicht auf zulässige Bereiche oder gerade Adressen geprüft. Die Byte-Anzahl kann beliebig bestimmt werden, es wird jedoch immer in Longword-Schritten kopiert, was bei String-Puffern ggfs. dazu führen kann, daß der Backtrailer überschrieben wird und das System abstürzt. Die Angabe von 14 oder 16 Bytes ist gleichbedeutend, da 14 nicht durch 4 teilbar ist.

```
Dim Reg%(16)                    ! DIM RCALL-Register
Rout$=Mkl$($60000006)           ! bra      (PC)+6
Rout$=Rout$+Mki$($22D8)         ! move.l   (a0)+1,(a1)+1
Rout$=Rout$+Mki$($5881)         ! addq.l      #4,d1
Rout$=Rout$+Mki$($B280)         ! cmp.l       d0,d1
Rout$=Rout$+Mkl$($6D00FFF8)     ! blt      (PC)-8
Rout$=Rout$+Mki$($4E75)         ! rts
~Fre(0)                         ! Garbage Collection
Sta%=V:Rout$                    ! Startadresse der Routine
Pbox 100,100,220,120            ! Grafik ausgeben
A$=Space$(32000)                ! Zielpuffer einrichten
Reg%(0)=32000                   ! Byte-Anzahl (durch 4 teilbar)
Reg%(1)=0                       ! Zähler d1 löschen
Reg%(8)=Xbios(2)                ! Quelladresse (gerade)
Reg%(9)=V:A$                    ! Zieladresse (gerade)
Rcall Sta%,Reg%()               ! RCALL-Aufruf
Cls                             ! Bildschirm löschen
Sput A$                         ! Zielpuffer zeigen
```

Es ist übrigens interessant, daß die Routine nicht funktioniert bzw. sogar abstürzen kann, wenn die DIM-Anweisung im Programm hinter der Zeile Sta%=V:Rout$ stehen würde. Das hat den Grund, daß durch DIMs die Adressenlage von String-Variablen und Feldern verschoben werden kann, so daß ein vorher ermittelter VARPTR ins Leere zeigt. Grundsätzlich ist es daher ratsam, VARPTR-Aufrufe unmittelbar vor der Adressverwendung einzusetzen (evtl. vorher FRE(0) aufrufen).

Beachten Sie auch, daß statt der V2.xx-Hexwertangabe &H in V3.0 auch $ verwendet werden kann. So angegebene Hexawerte werden

vom Interpreter in &H umgewandelt. Ebenso verhält es sich mit Binärwerten. Statt der V2.xx-Variante &X kann hier % eingesetzt werden.

7. Textoperationen

7.1 String-Manipulationen

MID$()= Teil-String zuweisen
In V3.0: { MI)= }

MID$(Ziel$,Start [,Anz])="Text"

Ziel$ gibt eine String-Variable an, in die "Text" ab Start eingesetzt werden soll. Anz gibt optional die Anzahl der Zeichen von Text an, die maximal in Ziel$ eingesetzt werden sollen. Die ursprüngliche Länge von Ziel$ bleibt unverändert. Es werden max. soviele Zeichen von Text eingefügt, wie ab Start in Ziel$ hineinpassen.

Beispiele hierzu finden Sie unter FORM INPUT und FUNCTION.

LSET { LS } Zeichen(kette) linksbündig einsetzen

LSET Ziel$="Text"

Text kann eine beliebige Zeichenkette oder String-Variable sein, deren Inhalt linksbündig in Ziel$ eingefügt wird. Dabei bleibt die ursprüngliche Länge von Ziel$ unverändert. Ist Ziel$ kürzer als Text, wird Text bei der Länge von Ziel$ abgeschnitten. Ist Ziel dagegen länger als Text, werden die restlichen Stellen von Ziel$ mit Leerzeichen aufgefüllt.

Ein Beispiel zu LSET finden Sie in Kap. 5.5.1 "Funktionsweise einer Random-Access-Datei".

RSET { RS } Zeichen(kette) rechtsbündig einsetzen

RSET Ziel$="Text"

RSET ist grundsätzlich mit LSET vergleichbar. Der einzige Unterschied besteht darin, daß "Text" am Ende des Ziel-Strings eingesetzt wird. Beide Befehle finden überwiegend in Random-Acces-Dateien Verwendung, wo es darum geht, daß trotz String-Veränderung die Längen der Eintragszeilen unverändert erhalten bleiben.

7.2 String-Analyse

INSTR() — Zeichen(kette) in einem String suchen

Var=INSTR(Ziel$,Such$ [,Start])
Var=INSTR([Start,] Ziel$,Such$)

Liefert einen Wert, der die absolute Position von Such$ in Ziel$ angibt. Ist Such$ in Ziel$ nicht enthalten, wird der Wert 0 zurückgegeben. Sind beide Strings leer, erhält man eine 1. Bei Verwendung der Option Start wird ab der damit angegebenen Zeichenposition gesucht. Bei der Angabe des Such-Strings ist darauf zu achten, daß hier zwischen Groß- und Kleinschreibung unterschieden wird.

Beispiel:

```
Ziel$="Kaum zu glauben. Ich trau' meinen Augen nicht."
Such$="au"
@Astring(0,Ziel$,Such$,Ibk$)     ! V3.0-Aufruf
' @Astring(0,Ziel$,Such$,*Ibk$)  ! V2.xx-Aufruf
For I%=1 To Len(Ibk$) Step 2
  Print Cvi(Mid$(Ibk$,I%,2))
Next I%
Procedure Astring(S.pos%,Z.str$,S.str$,Var Bk$) ! V3.0-Kopf
  ' Procedure Astring(S.pos%,Z.str$,S.str$,Bk%) ! V2.xx-Kopf
  '
  ' Erweiterte INSTR-Routine, die alle (!) Positionen
  ' eines Strings innerhalb eines Ziel-Strings ab einer
  ' evtl. angegebenen Startposition ermittelt und die
  ' Positionsliste als MKI$-Daten-String zurückgibt.
  '
  ' S.pos% = String-Position, ab welcher gesucht werden soll.
  '          0 wird wie bei INSTR als 1 interpretiert.
  ' Z.str$ = Zu untersuchender Ziel-String
  ' S.str$ = Zu suchender String
  ' Bk$    = In V3.0 VAR-Variable ----.  ist nach Rückkehr
  '                                    |- entweder leer (String
  '                                    |  nicht gefunden) oder
  '                                    |  enthält der Reihe nach
  '                                    |  die gefundenen Positionen
  ' ' Bk%  = in V2.xx Pointer auf      |- als MKI$-Werte
  '          Rückgabe-String-Variable  -'
  Local X$,Pos%
  Repeat                               ! Such-Schleife >--------.
    Pos%=Instr(S.pos%,Z.str$,S.str$) ! Position feststellen    |
    If Pos%>0                          ! String enthalten?      |
      X$=X$+Mki$(Pos%)                 ! Position eintragen     |
      S.pos%=Pos%+1                    ! Startposition erhöhen  |
    Endif                              !                        |
  Until Pos%=0                         ! Kein String mehr?  <---'
  Bk$=X$                               ! V3.0-'MKI$'--. Positions-
  ' *Bk%=X$                            ! V2.xx-'MKI$'-'  Rückgabe
Return
```

Bei der ersten ausgelieferten Version des V3.0-BASIC vom 10.5.88 konnte man den Eindruck gewinnen, daß Pointer-Übergaben an Prozeduren nicht mehr vorgesehen waren. Die überarbeitete Version vom 15.6.88 erfüllt jedoch auch in diesem Punkt die Erwartungen, die man an ein GFA-BASIC stellen kann. Bisher sind mir bei der neuesten V3.0-Version keine syntaktischen Unverträglichkeiten zu den V2.xx-Versionen mehr aufgefallen.

Ab jetzt heißt es wieder: "Selbstverständlich" kann der V2.xx-Aufruf und die entsprechende Pointer-Rückgabe auch in V3.0 verwendet werden!

Weitere Beispiele finden Sie unter DATA in der Prozedur Rplc und unter IF..ENDIF.

LEFT$() — Linksbündigen Teil-String ermitteln

Var$=LEFT$(Ziel$ [,Anz])

Ohne die Option Anz wird das erste Zeichen des Strings Ziel$ geliefert. Bei Verwendung der Option Anz werden ab String-Anfang Anz Zeichen von Ziel$ geliefert. Ist Anz größer als die Länge von Ziel$, so wird Ziel$ komplett als Ergebnis zurückgegeben. Ist Ziel$ ohne Inhalt (""), wird ein Null-String geliefert.

Beispiele zu LEFT$() finden Sie unter PRINT und unter DATA in den Prozeduren Pcode, Scode und Rplc.

LEN() — String-Länge ermitteln

Var=LEN(Var$)

Liefert die Länge des angegebenen Strings Var$.

Beispiele zu LEN() finden Sie unter FORM INPUT, INKEY$(), LINE INPUT (Prozedure Showdat), INSTR und an vielen Stellen im Buch verteilt.

MID$() Beliebigen Teil-String ermitteln

Var$=MID$(Ziel$,Start [,Anz])

Es wird ein Teil-String von Ziel$ geliefert. Start gibt die Position in Ziel$ an, ab welcher gelesen werden soll. Wird die Option Anz verwendet, werden ab Start soviel Zeichen geliefert, wie in Anz angegeben sind. Wird Anz nicht verwendet oder ist der in Anz angegebene Wert größer als die Menge der ab Start verbleibenden Zeichen von Ziel$, so wird der gesamte Rest-String ab Start zurückgegeben. Ist Ziel$ ohne Inhalt, wird ein Null-String ("") geliefert.

Anwendungsbeispiele zu MID$() finden Sie unter anderem bei FORM INPUT, BLOAD (Prozedur Degload), FILES, FOR..NEXT (Prozedur Mirror) und RIGHT$().

Version 3.0

PRED() Nächstkleineres ASCII-Zeichen ermitteln

Var$=PRED(Expr$)

Liefert das nächstkleinere ASCII-Zeichen des ersten Zeichens von Expr$. Ist als Abkürzung für CHR$(ASC(Expr$)-1) einsetzbar.

Beispiel:

```
For I%=65 To 76
  Print "ASCII des Zeichens ";Chr$(I%);
  Print " minus 1 = ";Asc(Pred(Chr$(I%)));
  Print " = Zeichen ";Pred(Chr$(I%))
Next I%
```

RIGHT$() Rechtsbündigen Teil-String ermitteln

Var$=RIGHT$(Ziel$ [,Anz])

Ohne die Option Anz wird das letzte Zeichen des Strings Ziel$ geliefert. Bei Verwendung der Option Anz werden vom String-Ende Anz Zeichen von Ziel$ geliefert. Ist Anz größer als die Länge von Ziel$, so wird Ziel$ komplett als Ergebnis zurückgegeben. Ist Ziel$ ohne Inhalt (""), wird ein Null-String zurückgegeben.

Beispiel:

```
X$="Dies ist ein DEMO-String zur Vorführung der String-"
X$=X$+"Trennfunktion CUT. Im Rückgabe-String (hier: A$)"
X$=X$+"wird eine Liste von MKI$-Wertepaaren zurückgegeben"
X$=X$+", die der Reihe nach die Positionen und Länge aller "
X$=X$+"extrahierten Strings enthält."
@Cut(X$,", -:.)",20,*A$)        ! Aufruf
For I%=1 To Len(A$) Step 4      ! MKI$-String in 4er-Steps
  '                             ! durchgehen
  Spos%=Cvi(Mid$(A$,I%,2))      ! Teil-String-Position lesen
  Slen%=Cvi(Mid$(A$,I%+2,2))    ! Teil-Stringlänge lesen
  Print Mid$(X$,Spos%,Slen%)    ! Teil-String ausgeben
Next I%
'
Procedure Cut(C.str$,C.sgn$,C.brt%,C.vec%)
  ' Teilt einen vorgegebenen String in Teile mit einer
  ' vorgegebenen max. Länge, wobei eine Liste von
  ' Trennzeichen angegeben werden kann, die dann Priorität
  ' vor der Länge haben. Der Augangs-String bleibt unverändert.
  '
  ' C.str$  =  Zu teilender Vorgabe-String
  ' C.sgn$  =  Beliebige Liste von Trennzeichen. Die
  '            Teil-Strings enden jeweils mit dem ersten
  '            Zeichen hinter dem gefundenen Trennzeichen.
  '            Ist C.sgn$ leer (""), gilt für die
  '            Trennung ausschließlich C.brt%.
  ' C.brt%  =  Max. neue Zeilenbreite
  ' C.vec%  =  Pointer auf eine Rückgabe-String-Variable, die
  '            nach Abschluß eine Liste von MKI$-Wertepaaren
  '            enthält, die der Reihe nach Position und Länge
  '            der einzelnen Teil-Strings angeben.
  '
  Local C.dum$,Cd.vec$,C.pos%,C.a$,C.j%
  C.pos%=1                      ! Positions-Puffer +1
  Do
    C.a$=Left$(C.str$,Min(C.brt%,Len(C.str$))) ! Teil-String
    '                           ! auf Länge trimmen
    If Len(C.sgn$)>0            ! Trennzeichen vorhanden?
      For C.j%=Len(C.a$) Downto 1 ! Alle Zeichen durchgehen
        Exit if Instr(C.sgn$,Mid$(C.a$,C.j%,1)) ! Abbruch,
        '                       ! wenn Trennzeichen gefunden
      Next C.j%                 ! Nächstes Zeichen
    Endif
    If C.j%=0                   ! Kein Trennzeichen?
      C.j%=C.brt%               ! Längenvorgabe hat Priorität
    Endif
    C.dum$=Left$(C.a$,Min(C.brt%,C.j%)) ! Teil-String "cutten"
    C.str$=Right$(C.str$,Len(C.str$)-Min((C.brt%),Len(C.dum$)))
    '                           ! Ausgangs-String kürzen
    Cd.vec$=Cd.vec$+Mki$(C.pos%)+Mki$(Len(C.dum$))
    '                           ! MKI$-String bilden
    Add C.pos%,Len(C.dum$)      ! Neue Startposition
    Exit if Len(C.str$)<C.brt% ! Exit, wenn Rest < max. Breite
  Loop
  Cd.vec$=Cd.vec$+Mki$(C.pos%)+Mki$(Len(C.str$)) ! MKI$-Rest
  *C.vec%=Cd.vec$               ! MKI$-String zurückgeben
Return
```

Weitere Beispiele zu RIGHT$() finden Sie unter bei INKEY$(), LINE INPUT (Prozedur Showdat), BLOAD (Prozedur Degload), FSNEXT() (Prozedur Getdir), sowie unter WAVE.

Version 3.0

RINSTR() Zeichen(kette) in einem String rückwärts suchen

Var=RINSTR(Ziel$,Such$ [,Start])
Var=RINSTR([Start,] Ziel$,Such$)

Liefert einen Wert, der die erste gefundene Position von 'Such$' in Ziel$ angibt. Bei der Durchsuchung des Ziel-Strings wird im Gegensatz zu INSTR am String-Ende begonnen. Weiteres siehe INSTR.

Version 3.0

SUCC() Nächstgrößeres ASCII-Zeichen ermitteln

Var$=SUCC(Expr$)

Liefert das nächstgrößere ASCII-Zeichen des ersten Zeichens von Expr$. Ist als Abkürzung für CHR$(ASC(Expr$)+1) einsetzbar.

Beispiel:

```
For I%=65 To 78
  Print "ASCII des Zeichens ";Chr$(I%);
  Print " plus 1  = ";Asc(Succ(Chr$(I%)));
  Print " = Zeichen ";Succ(Chr$(I%))
Next I%
```

7.3 String-Formatierung

SPACE$() Leerzeichen-String bilden

Var$=SPACE$(Anz)

Es wird ein String aus Anz Leerzeichen gebildet und zurückgegeben.

Beispiele finden Sie unter anderem unter BLOAD (Prozedur Degload), CHAIN, FILES und BPUT#.

STRINGS$() — Mehrfach-Zeichenkette bilden

Var$=STRING$(Anz,"Text")
Var$=STRING$(Anz,Ascii)

Es wird ein String gebildet, der sich daraus ergibt, daß Text so oft verkettet wird, wie in Anz angegeben. Text kann auch als Variable oder String-Konstrukt angegeben werden.

Soll ein einzelnes Zeichen multipliziert werden, kann statt Text auch der ASCII-Wert des Zeichens verwendet werden. Die entstehende Zeichenkette darf nicht mehr als 32767 Zeichen enthalten (maximale Größe einer String-Variablen).

Beispiele finden Sie unter anderem unter PRINT, CHAIN, FSNEXT() (Prozedur Getdir) und CALL.

Version 3.0

TRIM$() — Space-Zeichen eliminieren

Var$=TRIM$(Ziel$)

Eliminiert alle Spaces (Leerzeichen), die am String-Anfang oder am String-Ende von Ziel$ stehen und liefert den verbleibenden String-Teil. Besteht der übergebene String Ziel$ nur aus Leerzeichen oder er ist ohne Inhalt, erhält man einen Leer-String ("") zurück.

Beispiel:

```
For I%=1 To 4
  Read A$
  Print "Vorher : ->";A$;"<-",
  Print "Nachher: ->";Trim$(A$);"<-"
Next I%
Data   String 1  ,   String 2,String    3,String 4
```

UPPER$() — Buchstabenumwandlung Klein => Groß

Var$=UPPER$(Ziel$)

Trifft UPPER$ beim Lesen von Ziel$ auf einen kleingeschriebenen Buchstaben (ASCII-Werte 97 bis 129), so wird dieser in den entsprechenden Großbuchstaben (ASCII-Werte 65 bis 90) umgewandelt. Kleingeschriebene Umlaute (ä, ö, ü) werden ebenfalls in Uppercase (Ä, Ö, Ü) gewandelt. Alle anderen Zeichen bleiben unverändert.

Beispiele finden Sie unter FSNEXT() (Prozedur Getdir) und im erstei Beispiel zu FUNCTION.

Die entstehende Zeichenkette darf nicht mehr als 32767 Zeichen ent halten (maximale Größe einer String-Variablen).

8. Arithmetik-Befehle

8.1 Operatoren

Arithmetische Operatoren:

^	Potenzieren
-	Negatives Vorzeichen
* /	Multiplikation und Division
DIV	Ganzzahldivision (Entfernen von Dezimalstellen)
MOD	Modulo-Berechnung (*Rest der Ganzzahldivision*)
+ -	Addition und Subtraktion

Vergleichsoperatoren:

=	gleich	<> oder ><	ungleich
>	größer als	>= oder =>	größer oder gleich
<	kleiner als	<= oder =<	kleiner oder gleich
==	ungefähr gleich (28 Bit-Vergleich)		

Logische Operatoren:

AND	Konjunktion: Das Ergebnis von AND ist wahr, wenn beide Argumente wahr sind.
OR	Disjunktion: Das Ergebnis von OR ist wahr, wenn eines der Argumente wahr ist.
XOR	Exclusives Oder: Das Ergebnis von XOR ist falsch, wenn die Argumente beide wahr oder beide falsch sind.
NOT	Negation: Vertauscht Wahrheitswerte ins Gegenteil.
IMP	Implikation: Die Folgerung IMP ist nur dann falsch, wenn aus etwas Wahrem etwas Falsches folgt.
EQV	Äquivalenz: Umkehrung zu XOR. Das Ergebnis ist falsch, wenn sich die beiden Argumente unterscheiden.

Prioritäten:

( )	Klammern (höchste Priorität)
^	Potenzierung
-	Negatives Vorzeichen
* /	Multiplikation und Division
DIV MOD	Ganzzahldivision und Modulo-Berechnung
+ -	Addition und Subtraktion
= > < == <> >= <=	Vergleichsoperatoren
NOT AND OR, XOR IMP EQV	Logische Operatoren (niedrigste Priorität)

8.2 Mathematische Operationen

ADD { AD } — Additionsbefehl

ADD Var,Wert

Addiert Wert zum Inhalt der numerischen Variablen Var und legt anschließend das Ergebnis in Var ab.

Wird in Version V3.0 in Var eine Integer-Variable angegeben, wird ein evtl. in Wert angegebener Realwert als Integerwert behandelt.

DEC — Dekrementierung

DEC Var

Vermindert den Wert der numerischen Variable Var um den Wert 1.

DIV — Divisionsbefehl

Abk. in V3.0: { DI }

DIV Var,Wert

Dividiert den Inhalt der numerischen Variablen Var durch Wert und legt anschließend das Ergebnis in Var ab. Beachten Sie bitte die V3.0-Anmerkung zum ADD-Befehl.

INC { IN } — Inkrementierung

INC Var

Erhöht den Wert der numerischen Variable Var um den Wert 1.

MUL { MU } — Multiplikationsbefehl

MUL Var,Wert

Multipliziert den Inhalt der numerischen Variablen Var mit Wert und legt anschließend das Ergebnis in Var ab. Beachten Sie bitte die V3.0-Anmerkung zum ADD-Befehl.

SUB { S } — Subtraktionsbefehl

In V3.0: { SU }

SUB Var,Wert

Subtrahiert Wert vom Inhalt der numerischen Variablen Var und legt anschließend das Ergebnis in Var ab. Beachten Sie bitte die V3.0-Anmerkung zum ADD-Befehl.

Version 3.0

ADD() — Integer-Additionsfunktion

Var=ADD(Wert1,Wert2)

Addiert Wert1 zu Wert2 und liefert dann das entsprechende Integer-Ergebnis.

Anmerkung: Im Gegensatz zu den ADD-, SUB-, MUL-, DIV-Befehlen können diese Funktionen - wie jede andere Funktion auch - in die Ausgabe, in Zuweisungen, Ausdrücke, Bedingungsabfragen etc. eingebunden werden. Sie stehen stellvertretend für das Ergebnis, das sie liefern sollen. Außerdem können hiermit generell nur integrierte Werte berechnet werden. Von ggfs. aus der Operation entstehenden Realwerten wird nur der Vorkomma-Anteil geliefert. Eine Rundung findet nicht statt.

Die Werte können als Konstante, Variable, Ausdruck oder Funktion angegeben werden und werden durch die Operation nicht verändert. Werden die Werte als Realwerte angegeben, werden sie vor Ausführung der Operation auf ihren Integeranteil reduziert. Beispiel:

```
Print Int(100/33)+Int(345*Int(12-Int(5.6))) Mod 38
```

entspricht:

```
Print Mod(Add(Div(100,33),Mul(345,Sub(12,5.6))),38)
```

Version 3.0

DIV() Integer-Divisionsfunktion

Var=DIV(Wert1,Wert2)

Dividiert Wert1 durch Wert2 und liefert das entsprechende Integer-Ergebnis. Beachten Sie bitte die Anmerkung zur ADD()-Funktion.

Version 3.0

MOD() Integer-Modula-Funktion

Var=MOD(Wert1,Wert2)

Berechnet den ganzzahligen Rest der Integer-Divison Wert1 durch Wert2 (Modulo-Berechnung) und liefert das entsprechende Integer-Ergebnis. Beachten Sie bitte die Anmerkung zur ADD()-Funktion.

Version 3.0

MUL() Integer-Multiplikationsfunktion

Var=MUL(Wert1,Wert2)

Multipliziert Wert1 mit Wert2 und liefert das entsprechende Integer-Ergebnis. Beachten Sie bitte die Anmerkung zur ADD()-Funktion.

Version 3.0

SUB() Integer-Subtraktionsfunktion

Var=SUB(Wert1,Wert2)

Subtrahiert Wert2 von Wert1 und liefert das entsprechende Integer-Ergebnis. Beachten Sie bitte die Anmerkung zur ADD()-Funktion.

8.3 Numerische Funktionen

ABS() Betrags-Funktion

Var=ABS(Arg)

ABS gibt das Argument Arg vorzeichenlos als positiven (absoluten) Wert zurück. Dieser Wert ist immer gleich oder größer Null.

EVEN() Zahl auf "gerade" testen

Var=EVEN(Arg)

Liefert -1, wenn Arg gerade ist. Sonst wird 0 geliefert.

EXP() Exponential-Funktion

Var=EXP(Arg)

Berechnet das Ergebnis des Exponenten Arg zur Basis e (Eulersche Zahl). Gleichbedeutend mit: 2.718281828462 ^ Arg. Arg muß auf jeden Fall größer Null sein, da sonst eine Fehlermeldung produziert wird. EXP ist die Umkehrfunktion zu LOG.

FIX() Ganzzahl-Funktion

Var=FIX(Arg)

Übergibt den ganzzahligen Anteil (Integer) der Real-Zahl Arg. Die Funktion rundet Zahlen weder auf noch ab, sondern entfernt nur die Dezimalstellen. Im Minusbereich wird dadurch der Minuswert **kleiner**. FIX(-12.33) ergibt -12. FIX(33.17) ergibt 33. FIX ist identisch mit TRUNC.

FRAC() Dezimalstellen-Funktion

Var=FRAC(Arg)

Liefert den Dezimalanteil (Nachkommastellen) von Arg, falls Arg eine reelle Zahl ist, bzw. den Wert 0, falls Arg integer ist.

INT() Ganzzahl-Funktion

Var=INT(Arg)

Wandelt die Zahl Arg in eine Integer-Zahl. Ist Arg ein Realwert, so wird die nächstkleinere Ganzzahl zurückgegeben. Im Gegensatz zu FIX() und TRUNC() wird dadurch im Minusbereich der Minuswert **größer**.

```
INT(-12.33) ergibt -13        INT(33.17) ergibt 33.
```

LOG() Logarithmus-Funktion

Var=LOG[10](Arg)

LOG() gibt den natürlichen (Neperschen) Logarithmus des in Klammern angegebenen numerischen Ausdrucks Arg zur Basis e zurück.

```
Basis e => 2.718281828 => Eulersche Zahl
```

Das Ergebnis von 2.718281828462^LOG(Arg) ergibt Arg.

LOG10() liefert dagegen den dekadischen (Briggsschen) Logarithmus des in Klammern übergebenen Arguments Arg zur Basis 10.

Das Ergebnis von 10^LOG10(Arg) ergibt Arg.

Arg muß auf jeden Fall größer Null sein, da sonst eine Fehlermeldung produziert wird. LOG() ist die Umkehrfunktion zu EXP().

ODD() Zahl auf "ungerade" testen

Var=ODD(Arg)

Liefert -1, wenn Arg ungerade ist. Sonst wird 0 geliefert.

Version 3.0

PRED() Nächstkleinere Ganzzahl ermitteln

Var=PRED(Arg)

Liefert die nächstkleinere Integerzahl vor Arg. Bei Arg als Realzahl werden vor Ausführung der Operation die Nachkommastellen integriert (siehe INT).

Version 3.0

ROUND() Rundungs-Funktion

Var=ROUND(Arg [,Stelle])

Rundet Arg mathematisch exakt auf eine ganze Zahl. Wird die Option Stelle verwendet, wird auf die angegebene Nachkommastelle gerundet. Ist Stelle negativ, wird auf die entsprechende Stelle vor dem Komma gerundet.

Sollen in den V2.xx-Versionen mathematisch exakte Rundungen vorgenommen werden, müssen Sie dazu die Funktion INT() verwenden. Addieren Sie dazu zum INT()-Argument den Wert 0.5. Das bedeutet, das die entstehende Integerzahl bei einem gebrochenen Anteil von < 0.5 abgerundet und bei einem gebrochenen Anteil von > 0.5 aufgerundet wird.

SGN() — Vorzeichen ermitteln

Var=SGN(Arg)

Liefert das Vorzeichen (engl.: SiGN) von Arg.

```
1 wenn Arg > 0/-1 wenn Arg < 0/0 wenn Arg = 0
```

SQR() — Wurzel-Funktion

Var=SQR(Arg)

Es wird die 2. Wurzel (Quadratwurzel; engl.: SQuare Radical) von Arg geliefert. Wird eine höhere Wurzel gebraucht, so ist diese über den Umweg der Potenzierung mit gebrochenem Exponenten zu berechnen:

```
3. Wurzel = Arg^(1/3)
4. Wurzel = Arg^(1/4)
   etc.
```

Arg muß auf jeden Fall gleich oder größer Null sein, da sonst eine Fehlermeldung produziert wird.

Version 3.0

SUCC() — Nächstgrößere Ganzzahl ermitteln

Var=SUCC(Arg)

Liefert die nächstgrößere Integerzahl nach Arg. Bei Arg als Realzahl werden die Nachkommastellen integriert (siehe INT).

TRUNC() — Ganzzahl-Funktion

Var=TRUNC(Arg)

TRUNC ist identisch mit FIX. Weiteres siehe dort.

8.4 Trigonometrische Funktionen

Version 3.0

ACOS() — Arcuscosinus-Funktion

Var=ACOS(Arg)

Berechnet den Arcuscosinus von Arg. Arg ist ein Winkel in Radian. Soll der Winkel in Grad eingegeben werden, muß dieser vorher mit PI/180 multipliziert bzw. durch DEG gewandelt werden.

Version 3.0

ASIN() — Arcussinus-Funktion

Var=ASIN(Arg)

Berechnet den Arcussinus von Arg. Weiteres siehe unter ACOS().

ATN() — Arcustangens-Funktion

Var=ATN(Arg)

Das Argument Arg ist ein Tangenswert, aus dem der ihm entsprechende Winkel in Radian berechnet wird. Es wird ein Wert zwischen -PI/2 und +PI/2 geliefert. Benötigt man die Winkelangabe in Grad, so muß das Ergebnis mit 180/PI multipliziert bzw. in V3.0 durch DEG gewandelt werden.

COS() — Cosinus-Funktion

Var=COS(Arg)

Berechnet den Cosinus von Arg. Arg ist ein Winkel in Radian. Soll der Winkel in Grad eingegeben werden, muß dieser vorher mit PI/180 multipliziert bzw. in V3.0 durch DEG gewandelt werden.

Beispiel:

```
For I=0 To 90 Step 15
  Print "Sinus   ";I;" GRAD : ";Sin(I*Pi/180)
  Print "Cosinus ";I;" GRAD : ";Cos(I*Pi/180)
  Print "Tangens ",I;" GRAD : ";Tan(I*Pi/180)
Next I
For I=-Pi+0.0000001 To Pi Step Pi/90
```

```
    Print "Sinus    ";I;"  RAD : ";Sin(I)
    Print "Cosinus ";I;"  RAD : ";Cos(I)
    Print "Tangens ";I;"  RAD : ";Tan(I)
Next I
```

Ein weiteres Beispiel zu COS() und SIN() finden Sie unter PI.

Version 3.0

COSQ() — Interpolierte Cosinus-Funktion mit Grad-Angabe

Var=COSQ(Grad)

Ermittelt den 16tel-Grad-interpolierten Cosinus des in Grad angegeben Winkels Grad (ca. 10 mal schneller als COS()).

Version 3.0

DEG() — Umwandlung in Grad

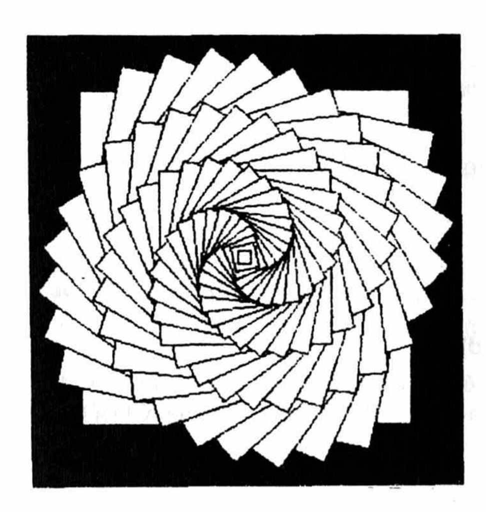
Var=DEG(Radian)

Rechnet die Radian-Winkelangabe in Gradmaß (DEGree) um. Entspricht dem Ergebnis von Arg*PI/180.

PI — Kreiszahl

PI

Reservierte Variable. Steht dort, wo sie eingesetzt wird für die konstante Kreiszahl PI (3.1415926536).



Beispiel (für Hires/Midres):

```
Yt%=Min(2,3-Xbios(4))              ! Y-Auflösungsteiler
Deffill ,2,4                       ! DEFFILL grau
For I%=0 To 360 Step 12            !---.
  @Cbox(2,320,200/Yt%,250,I%)         !- PBOX-Schleife
Next I%                            !---'
Cls                                ! Bildschirm klar
For I%=0 To 720 Step 12            !---.
  Add J%,3              ! Radius +3   !- BOX-Schleife
  @Cbox(1,320,200/Yt%,10+J%,I%)    !---'
Next I%
'
Procedure Cbox(Mod%,Xp%,Yp%,Rd%,Wi%)  ! (für V3.0 und V2.xx)
  ' Produziert ein Quadrat, das in einem beliebigen Winkel
  ' und mit beliebiger Größe dargestellt werden kann.
  ' Mod%    = Darstellungsmodus
  '            1 = BOX
  '            2 = PBOX
  '            3 = MARKER
  ' Xp%/Yp% = Koordinaten des Mittelpunktes (Drehpunktes)
  ' Rd%     = Umkreisradius
  ' Wi%     = Neigungswinkel
  '
  Local J%,I%,Yt%,Dg
  Yt%=Min(2,3-Xbios(4))           ! Y-Auflösungsteiler
  Erase Px%()                     ! POLY-X-Feld löschen
  Erase Py%()                     ! POLY-Y-Feld löschen
  Dim Px%(4),Py%(4)               ! POLY-Felder dimensionieren
  For I%=-Wi%+45 To -Wi%+360+45 Step 90 ! Einmal rundum
    Dg=I%*Pi/180                  ! In Grad umrechnen
    Px%(J%)=Xp%+(Sin(Dg)*Rd%*Sqr(2)/2+0.5)     !- Koordinaten..
    Py%(J%)=Yp%+(Cos(Dg)*Rd%/Yt%*Sqr(2)/2+0.5) !- ..berechnen
    '
    ' In V3.0 auch möglich:
    ' Px%(J%)=Xp%+(Sinq(I%)*Rd%*Sqr(2)/2+0.5)
    ' Py%(J%)=Yp%+(Cosq(I%)*Rd%/Yt%*Sqr(2)/2+0.5)
    '
```

```
    Inc J%                       ! Ecken-Zähler +1
  Next I%                        ! Nächste Ecke
  If Mod%=1                      ! BOX-Darstellung?
    Polyline 5,Px%(),Py%()       ! Dann Polyline
  Endif
  If Mod%=2                      ! PBOX-Darstellung?
    Polyfill 5,Px%(),Py%()       ! Dann Polyfill
  Endif
  If Mod%=3                      ! MARKER-Darstellung?
    Polymark 5,Px%(),Py%()       ! Dann Polymark
  Endif
Return
```

Version 3.0

RAD() — Umwandlung in Radian (Bogenmaß)

Var=RAD(Grad)

Rechnet die Grad-Winkelangabe Grad in Bogenmaß (RADian) um.

SIN() — Sinus-Funktion

Var=SIN(Arg)

Berechnet den Sinus von Arg. Weiteres siehe unter COS().

Version 3.0

SINQ() — Interpolierte Sinus-Funktion mit Grad-Angabe

Var=SINQ(Grad)

Ermittelt den 16tel-Grad-interpolierten Sinus des in Grad angegeben Winkels Grad (ca. 10 mal schneller als SIN()).

TAN() — Tangens-Funktion

Var=TAN(Arg)

Berechnet den Tangens von Arg. Weiteres siehe unter COS().

8.5 Vergleichsoperationen

MAX() — Größten Wert/String ermitteln

Var=MAX(Expr1,Expr2 [,Expr3,...])
Var$=MAX(Expr1$,Expr2$ [,Expr3$,...])

Gibt den größten Wert einer Werteliste bzw. den größten String einer String-Liste zurück. Expr kann eine beliebiger numerischer- oder Textausdruck, Wert, String, bzw. Variable sein. Alle Ausdrücke müssen demselben Typ angehören.

Bei String-Vergleichen wird der größte String ermittelt, indem der Reihe nach alle Einzelzeichen der zu vergleichenden Strings überprüft werden. Haben beide Zeichen denselben ASCII-Wert, werden solange die nächsten beiden Zeichen geprüft, bis Sie sich unterscheiden oder einer der beiden Strings keine Zeichen mehr enthält. Im ersten Fall ist der Ausdruck größer, dessen zuletzt geprüftes Zeichen den größeren ASCII- Wert besitzt. Im zweiten Falle ist es der String mit der größeren Länge.

Beispiele hierzu finden Sie unter anderem unter PRINT (Prozedur Pstep), FOR..NEXT (Prozedur Mirror) und unter IF..ENDIF.

MIN() — Kleinsten Wert/String ermitteln

Var=MIN(Expr1,Expr2 [,Expr3,...])
Var$=MIN(Expr1$,Expr2$ [,Expr3$,...])

Gibt den kleinsten Wert einer Werteliste bzw. den kleinsten String einer String-Liste zurück. Beispiele hierzu finden Sie unter anderem unter INPUT$, PRINT (Prozedur Pstep), BPUT#.

Sonst siehe Erläuterungen zu MAX().

8.6 Bit-Operationen

Version 3.0

AND() Konjunktions-Funktion

Var=AND(Wert1,Wert2)

Verknüpft logisch die beiden angegebenen Werte im AND-Modus und liefert das Integer-Ergebnis (siehe Kapitel 4 "Basis-BASIC").

Die Booleschen Funktionen stehen jeweils als Ersatz für die entsprechende logische Verknüpfungsoperation:

```
AND(x,y)      entspricht      (x AND y)
 OR(x,y)      entspricht      (x OR y)
XOR(x,y)      entspricht      (x XOR y)
EQV(x,y)      entspricht      (x EQV y)
IMP(x,y)      entspricht      (x IMP y)
```

Beispiel:

```
Print &X11101101 And &X1111'''''And(%11101101,%1111)
Print &HED Or &HF'''''Or($ED,$F)
Print 237 Xor 15'''''Xor(237,15)
```

Durch die Funktionsform werden diese Operationen für den Programmierer übersichtlicher und für das BASIC schneller "erfassbar", wodurch sich hier Geschwindigkeitsvorteile von bis zu 10 Prozent ergeben.

In der Beschreibung zu IF..ENDIF habe ich den Vorgang der Bedingungsstellung durch AND und OR zu erklären versucht. Eine häufige Bedingungsform ist z.B.:

```
IF ((v1%=w1 OR v2%<>w1) AND v3%>(ex1%+w2)) OR v4%<w2
```

Durch die Verknüpfungsfunktionen kann diese Bedingung folgendermaßen umgestaltet werden:

```
IF OR(AND(OR(v1%=w1,v2%<>w1),v3%>(ex1%+w2)),v4%<w2)
```

Durch die klare Klammersetzung kann ein Ausdruck dieser Art wesentlich leichter analysiert werden. Ein kleiner Nachteil ist, daß OR- oder AND-Ketten etwas unübersichtlicher werden.

Aus: `IF v1%>w1 OR v2%<>w1 OR v3%<>v1% OR v4%=w2`

wird: `IF OR(OR(OR(v1%>w1,v2%<>w1),v3%<>v1%),v4%=w2)`

Aber das ist wohl Geschmacksache. Man wird ja durch nichts daran gehindert, in solchen Fällen die erste Variante zu verwenden.

Version 3.0

BCHG() Einzel-Bit wechseln (Xor-en)

Var=BCHG(Wert,Bit)

Wechselt das angegebene Bit von Wert (Wert XOR 2^ Bit). War das Bit vorher 1, ist es anschließend 0 und umgekehrt.

Beachten Sie bei den Einzel-Bit-Funktionen, daß die Bit-Zählung von rechts mit der Bit-Nummer 0 beginnt. Bit kann beliebig angegeben werden, wobei größere Werte als 31 (max. Breite=32 Bit) nicht mehr korrekt verarbeitet werden. Größere Angaben für Wert als 2^32-1 (&HFFFFFFFF = max. Longwert) werden nicht verarbeitet.

Insbesondere sind diese Funktionen für die Verarbeitung von Bit-Vektoren geeignet, wie sie vielfach bei System-Funktionen (AES, GEMDOS, BIOS und XBIOS) verwendet werden.

Um z.B. die XBIOS-Funktion 33 (SETPTR, siehe unter XBIOS) auszuwerten, kann man folgendermaßen vorgehen:

```
If Btst(Xbios(33,-1),5)    ! Bit 5 gesetzt?
  Void Xbios(33,Bchg(Xbios(33,-1),5))    ! Dann XOR
  ' Oder:
  ' Void Xbios(33,Bclr(Xbios(33,-1),5)) ! Dann aus
Endif
```

Dieses Beispiel stellt zuerst fest, ob das Bit 5 (6. von rechts) der aktuellen Drucker-Einstellung gesetzt ist. Ist es das, so wird das Bit "umgeschaltet", so daß anschließend der Drucker statt auf Endlospapier auf Einzelblattzufuhr eingestellt ist. Statt der BCHG()-Funktion könnte man hier ebensogut BCLR() einsetzen.

Soll das Bit wieder gesetzt werden (also wieder Endlospapier einschalten), sähe das Ganze so aus:

```
If Btst(Xbios(33,-1),5)=False  ! Bit 5 nicht gesetzt?
  Void Xbios(33,Bchg(Xbios(33,-1),5))    ! Dann XOR
  ' Oder:
```

```
  ' Void Xbios(33,Bset(Xbios(33,-1),5)) ! Dann an
Endif
```

Auch hier ließe sich die BCHG()-Funktion problemlos gegen BSET() austauschen. Grundsätzlich könnte die IF-Abfrage bei diesen beiden Beispielen wegfallen, wenn in beiden Fällen die jeweils 2. Variante (in der REM-Zeile) verwendet würde, da das Bit dann unabhängig vom vorherigen Zustand sowieso aus- bzw. eingeschaltet worden wäre. Die IF-Abfrage ist hier nur für die BCHG()-Funktion erforderlich.

BCHG() als V2.xx-Funktion:

```
Print Bin$(&X11010001)''Bin$(@Bchg(&X11010001,4))
Deffn Bchg(Wert%,Bit%)=Wert% Xor 2^Bit%
```

Version 3.0

BCLR() — Einzel-Bit löschen

Var=BCLR(Wert,Bit)

Löscht das angegebene Bit von Wert (Wert AND NOT 2^Bit). War das Bit vorher 1 ist es anschließend 0. 0 bleibt 0. Weiteres siehe unter BCHG().

Als V2.xx-Funktion:

```
Print Bin$(&X11010001)''Bin$(@Bclr(&X11010001,4))
Deffn Bclr(Wert%,Bit%)=Wert% And Not 2^Bit%
```

Version 3.0

BSET() — Einzel-Bit setzen

Var=BSET(Wert,Bit)

Setzt das angegebene Bit von Wert (Wert OR 2^Bit). War das Bit vorher 0 ist es anschließend 1. War es 1, bleibt es 1. Weiteres siehe unter BCHG().

BSET() als V2.xx-Funktion:

```
Print Bin$(&X11010001)''Bin$(@Bset(&X11010001,3))
Deffn Bset(Wert%,Bit%)=Wert% Or 2^Bit%
```

Version 3.0

BTST() Einzel-Bit auf an/aus testen

Var=BTST(Wert,Bit)

Testet das angegebene Bit von Wert (-SGN(Wert AND 2^Bit)). Ist das Bit gesetzt, wird eine -1 (TRUE) geliefert, sonst 0 (FALSE). Weiteres siehe unter BCHG().

BTST() als V2.xx-Funktion:

```
Print @Btst(&X11010001,4)
Deffn Btst(Wert%,Bit%)=(Wert% And 2^Bit%)<>0
```

Version 3.0

BYTE() Vorzeichenloses LO-Byte eines Wertes liefern

Var=BYTE(Wert)

Liefert vorzeichenlos die untersten 8 Bit von Wert.

BYTE() als V2.xx-Funktion:

```
Print Bin$(@Byte(&X1001011111010001))
Deffn Byte(Wert%)=Wert% And &HFF
```

Version 3.0

CARD() Vorzeichenloses LO-Word eines Wertes liefern

Var=CARD(Wert)

Liefert vorzeichenlos die untersten 16 Bit von Wert.

CARD() als V2.xx-Funktion:

```
Print Bin$(@Card(&X1011010001011111010001))
Deffn Card(Wert%)=Wert% And &HFFFF
```

Version 3.0

EQV() — Äquivalenz-Funktion

Var=EQV(Wert1,Wert2)

Verknüpft logisch die beiden angegebenen Werte im EQV-Modus und liefert das Integer-Ergebnis. Weiteres siehe unter AND().

EQV() als V2.xx-Funktion:

```
Print Bin$(@Eqv(&X11010001,&X10011100))
Deffn Eqv(Wert1%,Wert2%)=Wert1% Eqv Wert2%
```

Version 3.0

IMP() — Implikations-Funktion

Var=IMP(Wert1,Wert2)

Verknüpft logisch die beiden angegebenen Werte im IMP-Modus und liefert das Integer-Ergebnis. Weiteres siehe unter AND().

IMP() als V2.xx-Funktion:

```
Print Bin$(@Imp(&X11010001,&X10011100))
Deffn Eqv(Wert1%,Wert2%)=Wert1% Eqv Wert2%
```

Version 3.0

OR() — Disjunktions-Funktion

Var=OR(Wert1,Wert2)

Verknüpft logisch die beiden angegebenen Werte im OR-Modus und liefert das Integer-Ergebnis. Weiteres siehe unter AND().

OR() als V2.xx-Funktion:

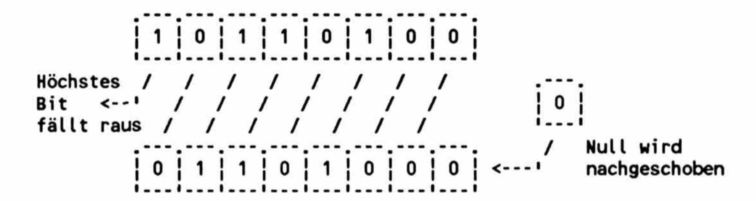
```
Print Bin$(@Or(&X11010001,&X10011100))
Deffn Or(Wert1%,Wert2%)=Wert1% Or Wert2%
```

Version 3.0

SHL() Bits links verschieben

Var=SHL(Wert,Bits) **Long-Shift-Left**
Var=SHL&(Wert,Bits) **Word-Shift-Left**
Var=SHL|(Wert,Bits) **Byte-Shift-Left**

Verschiebt den Inhalt von Wert um die Anzahl Bits nach links. Je verschobenem Bit wird Wert dabei mit 2 multipliziert (Integer-Multiplikation: Var=Wert*2^Bits). Das rechts freiwerdende Bit wird mit 0 gefüllt.



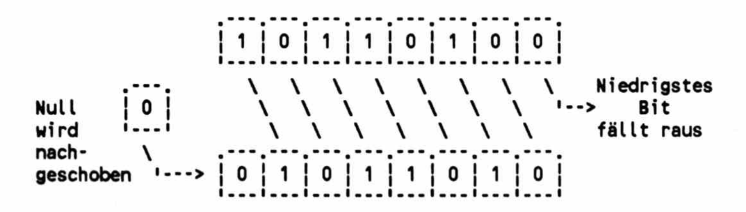
Bei Angabe von & hinter SHL werden nur die ersten 16 Bit (LO-Word) von Wert geshiftet, bei | nur die ersten 8 Bit (LO-Byte). Ist Wert bei SHL|() größer als 8 Bit, so werden als Ergebnis trotzdem nur die untersten 8 Bit der Operation geliefert. Dasselbe gilt für SHL&() (16 Bit), wobei dann jedoch das Bit 15 des Ergebnisses in die Bits 16-31 kopiert wird (vorzeichenbehaftet).

Version 3.0

SHR() Bits logisch rechts verschieben

Var=SHR(Wert,Bits)Long-Shift-Right
Var=SHR&(Wert,Bits) **Word-Shift-Right**
Var=SHR|(Wert,Bits) **Byte-Shift-Right**

Verschiebt den Inhalt von Wert um die Anzahl Bits nach rechts. Je verschobenem Bit wird Wert dabei durch 2 dividiert (Integer-Division: Var=Wert DIV 2^Bits). Das links freiwerdende Bit wird mit 0 gefüllt. Bei Komplementwerten bleibt also das Vorzeichen nicht erhalten.



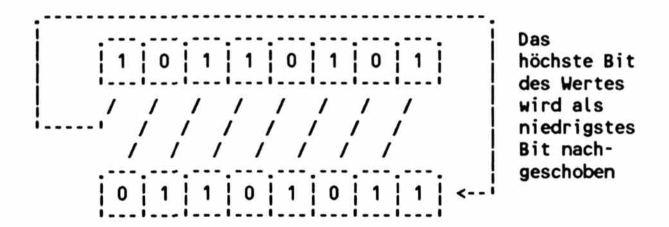
Bei Angabe von & hinter SHR werden nur die ersten 16 Bit (LO-Word) von Wert geshiftet, bei | nur die ersten 8 Bit (LO-Byte). Ist Wert bei SHR|() größer als 8 Bit, so werden als Ergebnis trotzdem nur die untersten 8 Bit der Operation geliefert. Dasselbe gilt für SHR&() (16 Bit), wobei dann jedoch das Bit 15 des Ergebnisses in die Bits 16-31 kopiert wird (vorzeichenbehaftet).

Version 3.0

ROL() — Bits links rotieren

Var=ROL(Wert,Bits) — **Long-Rotate-Left**
Var=ROL&(Wert,Bits) — **Word-Rotate-Left**
Var=ROL|(Wert,Bits) — **Byte-Rotate-Left**

Rotiert den Inhalt von Wert um die Anzahl Bits nach links. Das jeweils rechts freiwerdende Bit wird dabei mit dem jeweils links herausgeschobenen Bit gefüllt.



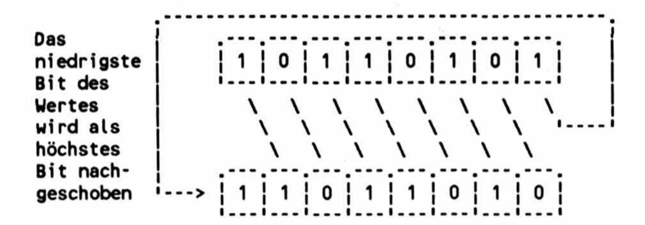
Bei Angabe von & hinter ROL werden nur die ersten 16 Bit (LO-Word) von Wert rotiert, bei | nur die ersten 8 Bit (LO-Byte). Ist Wert bei ROL|() größer als 8 Bit, so werden als Ergebnis trotzdem nur die untersten 8 Bit der Operation geliefert. Dasselbe gilt für ROL&() (16 Bit), wobei dann jedoch das Bit 15 des Ergebnisses in die Bits 16-31 kopiert wird (vorzeichenbehaftet).

Version 3.0

ROR() **Bits rechts rotieren**

Var=ROR(Wert,Bits) **Long-Rotate-Right**
Var=ROR&(Wert,Bits) **Word-Rotate-Right**
Var=ROR|(Wert,Bits) **Byte-Rotate-Right**

Rotiert den Inhalt von Wert um die Anzahl Bits nach rechts. Das jeweils links freiwerdende Bit wird dabei mit dem jeweils rechts herausgeschobenen Bit gefüllt.



Bei Angabe von & hinter ROR werden nur die ersten 16 Bit (LO-Word) von Wert rotiert, bei | nur die ersten 8 Bit (LO-Byte). Ist Wert bei ROR|() größer als 8 Bit, so werden als Ergebnis trotzdem nur die untersten 8 Bit der Operation geliefert. Dasselbe gilt für ROR&() (16 Bit), wobei dann jedoch das Bit 15 des Ergebnisses in die Bits 16-31 kopiert wird (vorzeichenbehaftet).

Version 3.0

SWAP() **HI- und LO-Word vertauschen**

Var=SWAP(Wert)

- Funktion - (siehe SWAP als Befehl)

Vertauscht das LO-Word (Bits 0-15) von Wert mit dessen HI-Word (Bits 16-31). Wert wird dabei grundsätzlich als 32-Bit-Integerwert interpretiert. Wird Wert als Realwert übergeben, werden evtl. vorhandene Nachkommastellen vor Ausführung der Operation integriert (siehe INT()). Kleinere Wert-Formate als 32 Bit werden auf Long erweitert.

SWAP() als V2.xx-Funktion:

```
Print Bin$(@Swap(&X10011110100010011101011001))
Deffn Swap(Wert%)=(Wert% And &HFFFF)*2^16+(Wert%/2^16)
```

Version 3.0

WORD() — Wert auf 32 Bit erweitern

Var=WORD(Wert)

Erweitert Wert arithmetisch (vorzeichenbehaftet) auf 32 Bit. Das Bit 15 von Wert wird dabei in die obersten 16 Bits des Ergebnisses kopiert. Ist also Bit 15 (16. von links) von Wert gesetzt, so ist das Ergebnis von WORD(Wert) negativ.

WORD() als V2.xx-Funktion:

```
Print Bin$(@Word(&X1111110111011001))
Deffn Word(Wert%)=((Wert% And 2^15)/2^15)*2^16*(2^15-1)-2^31+Wert%
```

Version 3.0

XOR() — eXclusivOR-Funktion

Var=XOR(Wert1,Wert2)

Verknüpft logisch die beiden angegebenen Werte im XOR-Modus und liefert das Integer-Ergebnis. Weiteres siehe unter AND().

XOR() als V2.xx-Funktion:

```
Print @Xor(&X11010001,&X10011100)
Deffn Xor(Wert1%,Wert2%)=Wert1% Xor Wert2%
```

8.7 Zufallswert-Erzeugung

Version 3.0

RAND() — 16-Bit-Integer-Zufallszahl

Var=RAND(n)

Übergibt eine vorzeichenlose 16-Bit-Integer-Zufallszahl aus dem Zahlenbereich 0 (inkl.) und n (exkl.). Größere Werte für n als 65535 werden durch n MOD 65535 auf den zulässigen Bereich umgerechnet.

RANDOM() 32-Bit-Integer-Zufallszahl

Var=RANDOM(n)

Übergibt eine 32-Bit-Integer-Zufallszahl aus dem Integer-Zahlenbereich 0 (inkl.) und n (exkl.), wobei n auch negativ sein kann.

Version 3.0

RANDOMIZE { RA } Zufallszahlengenerator initialisieren

RANDOMIZE [(Start)]

Inistialisiert den ST-Zufallszahlengenerator. Bei Verwendung des optionalen Parameters Start wird der Generator mit diesem Wert gestartet. Bei mehrfacher Verwendung desselben Startwertes beginnt immer dieselbe Zufallszahlenfolge mit Start als erstem Wert. Möchten Sie denselben Startwert intialisieren, der bei Systemstart gültig war, so kann die optionale Start-Klammer weggelassen oder RANDOMIZE 0 verwendet werden.

RND() Dezimalstellen-Zufallszahl

Var=RND [(Arg)]

Es wird ein 11stelliger (V2.xx) bzw. 13stelliger (V3.0) Zufallswert im Bereich zwischen 0 (inkl.) und 1 (exkl.) geliefert. Die gesamte Klammer ist optional und kann vernachlässigt werden. Wenn sie verwendet wird, so gilt Arg als Scheinargument ohne Bedeutung.

Beispiel:

```
Print Rnd(0)*10+3    ! Liefert zufällige Realzahl
                     ! im Bereich von 3 bis 10
```

9. Grafik

9.1 Grafikdefinitionen

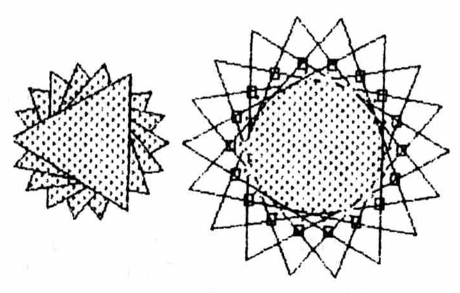
Version 3.0

BOUNDARY { BOU } **P-Grafikumrandung an/aus**

BOUNDARY Flag

Schaltet die Umrandung von P-Grafikobjekten (PBOX, PCIRCLE, POLYFILL etc.) an (Flag = 1) oder aus (Flag = 0).

Beispiel: Diese Prozedur eignet sich zur Erzeugung der unterschiedlichsten geometrischen Figuren. Spielen Sie ein wenig mit den Einstellungsparametern und den Punktangaben der POLYxxx-Befehle. Das Koordinatenfeld wurde vorsorglich auf maximal 60 Punkte dimensioniert, um Veränderungen der Schrittweite zulassen zu können. Ein rechtwinkliges Dreieck läßt sich z.B. dadurch erzeugen, indem Sie die Schrittweite STEP von 120 auf 90 ändern. Ein exzentrisches Dreieck entsteht, wenn Sie STEP kleiner 90 werden lassen. Ein gleichmäßiges Sechseck entsteht, indem Sie STEP 60 einstellen und gleichzeitig die Punktangaben bei POLYLINE auf 7 sowie bei POLYFILL und POLYMARK auf 6 Punkte ändern.



```
Defmark ,4                      ! Marker einstellen
Deffill ,2,4                    ! DEFILL grau
For I%=0 To 360 Step 20         ! Einmal rundherum in 20-Grad-Steps
  Boundary 0                    ! Rahmen aus
  @Ctri(1,120,110,100,I%)
  @Ctri(2,120,110,60,I%)
  @Ctri(3,120,110,64,I%)
  Boundary 1                    ! Rahmen an
  @Ctri(2,120,280,60,I%)
Next I%
'
```

```
Procedure Ctri(Mod%,Xp%,Yp%,Rd%,Wi%) ! Für Hires und Midres
  ' Produziert ein Dreieck, das in einem beliebigen Winkel
  ' und mit beliebiger Größe dargestellt werden kann.
  ' Mod%    = Darstellungsmodus
  '           1 = Nur Linie
  '           2 = Gefüllte Dreiecksfläche
  '           3 = Marker
  ' Möchten Sie nur die Eck-Koordinaten ermitteln, ohne daß
  ' die Figur gezeichnet wird, so übergeben Sie Mod% = 0.
  ' Die Eck-Koordinaten können Sie dann nach Rückkehr aus
  ' Px%(0)/Py%(0) bis Px%(2)/Py%(2) auslesen.
  '
  ' Xp%/Yp% = Koordinaten des Mittelpunktes (Drehpunktes)
  ' Rd%     = Umkreisradius
  ' Wi%     = Neigungswinkel
  '
  Local J%,I%,Yt%               ! Lokale Variablen
  Yt%=Min(2,3-Xbios(4))         ! Y-Auflösungsteiler
  Erase Px%()                   ! POLY-X-Feld löschen
  Erase Py%()                   ! POLY-Y-Feld löschen
  Dim Px%(4),Py%(4)             ! POLY-Felder dimensionieren
  For I%=-Wi% To -Wi%+360 Step 120 ! Einmal rundum
    Px%(J%)=Xp%+(Sin(I%*Pi/180)*Rd%+0.5)     ! Koordinaten...
    Py%(J%)=Yp%+(Cos(I%*Pi/180)*Rd%/Yt%+0.5) ! ...berechnen
    Inc J%                      ! Ecken-Zähler +1
  Next I%                       ! Nächste Ecke
  If Mod%=1                     ! Nur Linie?
    Polyline 4,Px%(),Py%()      ! Dann POLYLINE
  Endif
  If Mod%=2                     ! Gefüllte Figur?
    Polyfill 3,Px%(),Py%()      ! Dann POLYFILL
  Endif
  If Mod%=3                     ! Als Marker?
    Polymark 3,Px%(),Py%()      ! Dann POLYMARK
  Endif
Return
```

COLOR { CO } — Linienfarbe bestimmen

In V3.0: { C }

COLOR Farbe

Bestimmt das Farb-Register aus dem linien- und punktezeichnende Grafikobjekte (LINE, PLOT, DRAW, CIRCLE, BOX etc.) ihre Farben beziehen.

```
Hires  (640/400) =>  Farbe  = 0 oder 1
                     weiß, schwarz
Midres (640/200) =>  Farbe  = 0 .... 3
                     weiß, schwarz, rot, grün
```

```
Lowres (320/200) =>  Farbe = 0 ... 15
                     weiß, schwarz, rot, grün, blau
                     türkis, gelb, violett, hellgrau
                     dunkelgrau, hellrot, hellgrün
                     hellblau, helltürkis, hellgelb
                     hellviolett
```

Die unter den Auflösungen aufgeführten Farben sind jene, die bei Systemstart standardmäßig voreingestellt sind. Der Inhalt der Register kann mit dem Befehl SETCOLOR bestimmt werden.

Anmerkung: Wundern Sie sich bitte nicht, wenn der bei SETCOLOR verwendete Registerindex nicht mit dem COLOR-Registerindex übereinstimmt. Haben Sie mit SETCOLOR z.B. das Register 4 auf blau eingestellt und versuchen nun eine blaue Linie zu ziehen, indem Sie vor dem LINE-Befehl COLOR 4 einstellen, kann es sein, daß die Linie nicht blau, sondern orange (oder so) wird. Das liegt daran, daß mit SETCOLOR der TOS-Index angesprochen wird, während der COLOR-Befehl über das VDI realisiert wird.

Decodierungstabelle (Lowres):

SETCOLOR:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
COLOR :	0	2	3	6	4	7	5	8	9	10	11	14	12	15	13	1

COLOR :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SETCOLOR:	0	15	1	2	4	6	3	5	7	8	9	10	12	14	11	13

Decodierungstabelle (Midres):

SETCOLOR:	0	1	2	3
COLOR :	0	2	3	1

COLOR :	0	1	2	3
SETCOLOR:	0	3	1	2

Haben Sie z.B. mit SETCOLOR das Register 3 eingestellt, müssen Sie COLOR 6 einsetzen, um die Farbe aus dem SETCOLOR-Register 3 als Linienfarbe verwenden zu können. Anders herum: Haben Sie COLOR 14 verfügt, werden anschließend linienzeichnende Befehle mit der im SETCOLOR-Register 11 eingestellten Farbe ausgeführt.

Ein Beispiel finden Sie unter SETCOLOR.

DEFFILL { DEFF } — Füllmuster bestimmen

DEFFILL [Farbe],[Stil],[Muster]
DEFFILL [Farbe],Muster$

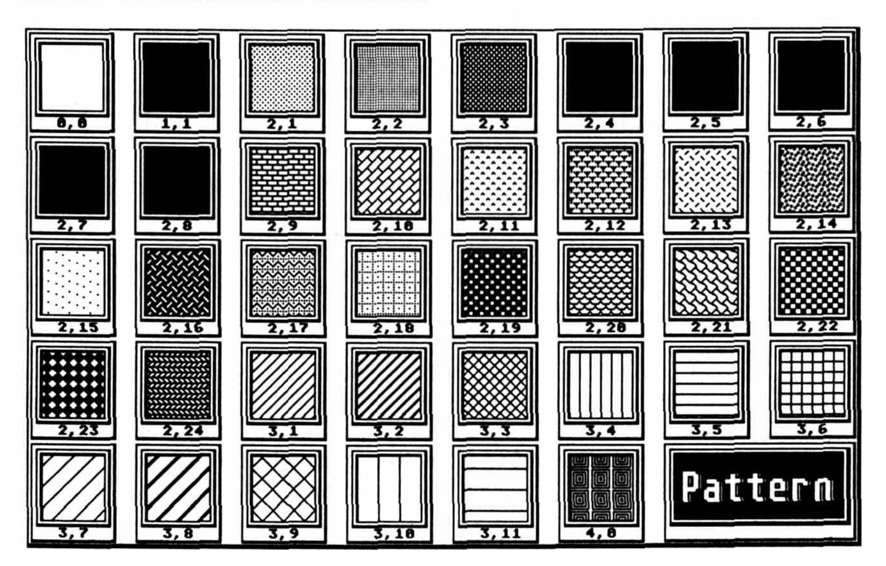
Legt Füllfarbe, Füllstil und Füllmuster für P-Grafikbefehle und FILL fest, bzw. ermöglicht eigene Muster-Definitionen.

Parameter Farbe siehe unter COLOR.

Stil		Muster
0 = Hintergrundfarbe	=>	Entfällt
1 = Objektfarbe	=>	Entfällt
2 = Punktiert	=>	2 bis 24
3 = Schraffiert	=>	1 bis 12
4 = Selbstdefiniert	=>	Atari- bzw. Benutzer-Muster

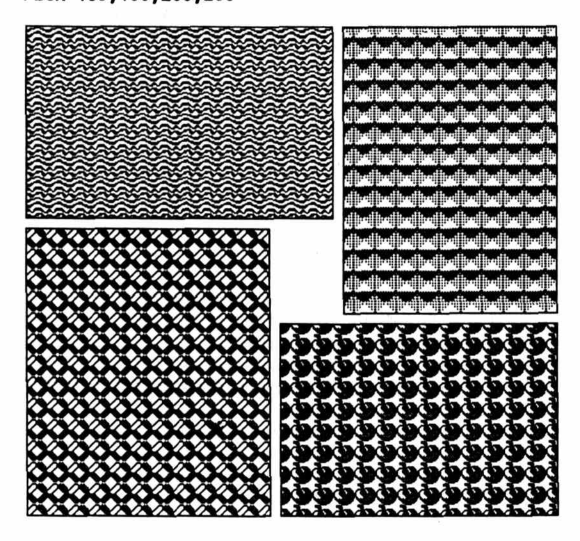
Parameter, die unverändert bleiben sollen, können ausgelassen werden (jedoch nicht die dazugehörigen Trennkommas).

Mit der Variante DEFFILL Farbe,Muster$ läßt sich ein eigenes Füllmuster einrichten. Dazu ist in Muster$ ein 16 (bzw. 32/64 siehe unten) Words langer String zu übergeben. Dieser beinhaltet 16 Words (bzw. 32/64 siehe unten) im MKI$-Format, die der Reihe nach die Bit-Muster der 16 Musterzeilen enthalten.



Beispiel:

```
F.muster$=Mki$(&X1111111111111111)
F.muster$=F.muster$+Mki$(&X1000000000000000)
F.muster$=F.muster$+Mki$(&X1000000110000000)
F.muster$=F.muster$+Mki$(&X1000001001000000)
F.muster$=F.muster$+Mki$(&X1000010000100000)
F.muster$=F.muster$+Mki$(&X1000100000010000)
F.muster$=F.muster$+Mki$(&X1001000110001000)
F.muster$=F.muster$+Mki$(&X1010001111000100)
F.muster$=F.muster$+Mki$(&X1001000110001000)
F.muster$=F.muster$+Mki$(&X1000100000010000)
F.muster$=F.muster$+Mki$(&X1000010000100000)
F.muster$=F.muster$+Mki$(&X1000001001000000)
F.muster$=F.muster$+Mki$(&X1000000110000000)
F.muster$=F.muster$+Mki$(&X1011111111111110)
F.muster$=F.muster$+Mki$(&X1001111111111000)
F.muster$=F.muster$+Mki$(&X1000000000000000)
Deffill 1,F.muster$
Pbox 100,100,200,200
```



In Version V3.0 kann für jede mögliche Bit-Plane jeweils ein 16 Word-Block in Muster$ übergeben werden. D.h., in den Words 1 bis 16 steht dann das Bit-Muster für die Plane 1 (Hires), in den Words 17 bis 32 das zusätzliche Bit-Muster für Plane 2 (Midres) und in den Words 33 bis 64 das zusätzliche Bit-Muster für Plane 3 und 4 (Lowres).

Beispiel 2:

```
Graphmode 2
For I%=0 To 7                !------.
  Deffill I%,3,I%                   ! Irgendein
  Pcircle 16,16,16                  ! Bit-Muster
  Deffill I%+2,3,I%+2               ! erzeugen
  Pcircle 48,16,16           !------'
Next I%
@Dfill3(7,7,Df$)             ! V3.0-Aufruf
' @Dfill2(7,7,*Df$)          ! V2.xx-Aufruf nur in Hires
Deffill 1,Df$                ! Neues Füllmuster..
Pbox 15,64,192,192           ! ...zeigen
Graphmode 3                  ! XOR-Modus
@Dfill3(42,7,Df$)            ! V3.0-Aufruf
' @Dfill2(42,7,*Df$)         ! V2.xx-Aufruf nur in Hires
Deffill 1,Df$                ! Neues Füllmuster..
Pbox 105,14,192,112          ! Zeigen (mit Überlappung)
'
Procedure Dfill3(D.x%,D.y%,Var D.ff$)   ! nur für V3.0
  ' Kopiert einen beliebigen 16*16-Bildschirmausschnitt im
  ' DEFFILL-Format in eine String-Variable (Hires/Midres/Lowres)
  ' D.x%  = Linke Quell-X-Koordinate
  ' D.y%  = Obere Quell-Y-Koordinate
  ' D.ff$ = VAR-String-Variable, welche nach Abschluß
  '         die Füllmusterdaten enthält
  Local D.fr$,Dc1%,Dc2%,D.f$,Xb%       ! Lokale Variablen
  Get D.x%,D.y%,D.x%+15,D.y%+15,D.fr$ ! Ausschnitt speichern
  Xb%=Xbios(4)                         ! Auflösung holen
  If Xb%=2                             ! Hires?
    D.f$=Right$(D.fr$,32)              ! Muster übertragen
  Else                                 ! Midres oder Lowres !
    For Dc1%=0 To 3-Xb%*2              !  2x    bzw.   4x
      For Dc2%=0 To 15                 ! 16 Musterzeilen...
        D.f$=D.f$+Mid$(D.fr$,7+Dc1%*2+Dc2%*(8-Xb%*4),2)
        ' ...isolieren und in String einbinden
      Next Dc2%                        ! Nächste Zeile
    Next Dc1%                          ! Nächste Bit-Plane
  Endif
  D.ff$=D.f$                           ! Muster-Rückgabe
Return
'
Procedure Dfill2(D.x%,D.y%,D.f%)       ! V2.xx (nur in Hires)
  ' D.x%, D.y% siehe oben
  ' D.f% = Pointer auf Rückgabe-String-Variable, welche
  '        nach Abschluß die Füllmusterdaten enthält
  Local D.fr$
  Get D.x%,D.y%,D.x%+15,D.y%+15,D.fr$
  *D.f%=Right$(D.fr$,32)
Return
```

DEFLINE { DE } Linien-Modi bestimmen

DEFLINE [Stil],[Dicke],[Form.a],[Form.e]

Mit diesem Befehl kann das bei linienzeichnenden Befehlen (BOX, LINE, CIRCLE, DRAW etc.) zu verwendende Linienmuster festgelegt werden.

```
Stil:
  0 = Linie in Hintergrundfarbe
  1 = Durchgezogene Linie
  2 = Gestrichelte Linie 1
  3 = Gepunktete Linie
  4 = Punkt-Strich-Linie
  5 = Gestrichelte Linie 2
  6 = Strich-Punkt-Punkt-Linie
 -1
 bis  = Benutzerstil
-32767
```

Der selbstdefinierte Linienstil setzt sich aus einem 15-Bit-Wert zusammen, wobei jedes gesetzte Bit einem Punkt in der Linie entspricht. Diese Zahl muß als Minuswert übergeben werden. Die Linie setzt sich dann aus dem Vielfachen dieser 15 Bits zusammen. Die Stil-Veränderungen wirken sich nur bei Liniendicke 1 aus. Alle anderen Liniendicken werden als Voll-Linie gezeichnet.

Dicke legt die Liniendicke fest, wobei diese nur in 2er-Schritten bis zum Maximalwert 39 (1, 3 .... 37, 39) verändert werden kann. Form.a und Form.e bestimmen die Linienanfangs- und -end-Form:

```
0 = Eckig
1 = Pfeilförmig
2 = Rund
```

Beispiel:

```
For I%=1 To 6
  Defline I%,1,0,0
  Line 200,55+I%*16,300,55+I%*16
  Print At(38,4+I%);I%
Next I%
Defline -18149,1,0,0         ! -18149 = -&X100011011100101
For I%=1 To 90 Step 4
```

```
  Box 10+I%,10+I%,190-I%,190-I%
Next I%
```

Über das VDI läßt sich auch das Linienmuster innerhalb von AES-Formularen (FILESELECT-, ALERT-Box etc.) variieren (siehe Beispiel 2 unter DEFTEXT).

DEFMARK { DEFM } — Markierungs-Symbol bestimmen

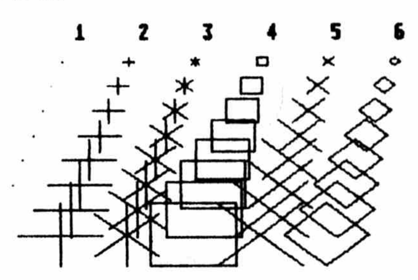
In V3.0: { DEFMA }

DEFMARK [Farbe],[Typ],[Größe]

Festlegung von Farbe, Typ und Größe der Markierungssymbole.

Markertyp:

```
1  =  Ein-Pixel-Punkt
2  =  Plus-Zeichen
3  =  Sechszackiger Stern
4  =  Liegendes Rechteck
5  =  Diagonalkreuz bzw. X
6  =  Liegende Raute
```



Alle Marker besitzen unveränderlich eine Liniendicke von einem Pixel. Die Markergröße (in Pixeln) ist nur in 11er-Schritten ab dem Wert 6 bis zum Maximalwert 83 veränderbar. Markerfarbe siehe unter COLOR.

Beispiel:

```
Dim X%(1),Y%(1)                 ! DIM POLYxxxx-Felder
For I%=1 To 6                   ! 6 Typen
  For J%=6 To 83 Step 11        ! Alle Größen
    X%(0)=16+I%*50-J%/1.5       ! X-Koordinate
    Y%(0)=20+J%*1.7             ! Y-Koordinate
    Defmark ,I%,J%              ! Marker einstellen..
    Polymark 1,X%(),Y%()        ! ...und zeichnen
  Next J%                       ! Nächste Größe
```

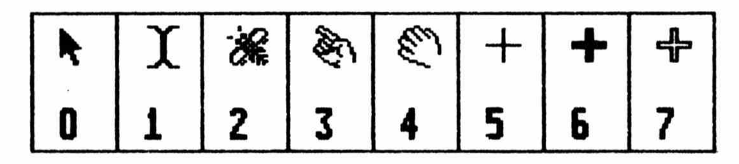
```
   Print At(4+I%*6,1);I%        ! Typen-Index ausgeben
Next I%                         ! Nächster Typ
```

DEFMOUSE { DEFMO } Mausform bestimmen

In V3.0: { DEFM }

DEFMOUSE Form
DEFMOUSE Maus$

Aufruf von selbstdefinierten oder systemeigenen Mausformen.



Form (vordefiniert):

```
0 = Pfeil              1 = X-Klammer (Text-Cursor)
2 = Biene              3 = Zeigende Hand
4 = Offene Hand        5 = Fadenkreuz fein
6 = Fadenkreuz grob    7 = Fadenkreuz umrandet
```

Statt Form kann durch Maus$ eine String-Variable angegeben werden, deren Inhalt im MKI$-Format die Mausform definiert.

Word 1
X-Koordinate des Aktionspunktes.

Word 2
Y-Koordinate des Aktionspunktes. Auf den Aktionspunkt werden anschließend alle Mausaktionen (z.B. Mousex, Mousey) bezogen.

Word 3
Immer MKI$(1).

Word 4
Maskenfarbe (Hintergrund des Mausbildes):

```
weiß = MKI$(0)/schwarz = MKI$(1)
```

Word 5
Cursor-Farbe (Mausbild).

Word 6 bis 21
16-Bit-Musterzeilen der Mausmaske.

Word 22 bis 37
16-Bit-Musterzeilen des Mausbildes.

Die Definition einer Mausform ist nur zweifarbig möglich.

Beispiel:

```
Deffill ,2,2                        ! DEFFILL hellgrau
Pbox 11,11,23,23                    ! Erstes Mausbild zeichnen
@Dmouse(10,10,0,0,Ms1$,Ms2$)        ! V3.0-Aufruf
' @Dmouse(10,10,0,0,*Ms1$,*Ms2$)    ! V2.xx-Aufruf
Line 17,12,17,22                    ! Mausbild verändern
@Dmouse(10,10,0,0,Ms3$,Ms4$)        ! V3.0-Aufruf
' @Dmouse(10,10,0,0,*Ms3$,*Ms4$)    ! V2.xx-Aufruf
Line 12,17,22,17                    ! Mausbild verändern
@Dmouse(10,10,0,0,Ms5$,Ms6$)        ! V3.0-Aufruf
' @Dmouse(10,10,0,0,*Ms5$,*Ms6$)    ! V2.xx-Aufruf
Circle 17,17,6                      ! Mausbild nochmals verändern
@Dmouse(10,10,0,0,Ms7$,Ms8$)        ! V3.0-Aufruf
' @Dmouse(10,10,0,0,*Ms7$,*Ms8$)    ! V2.xx-Aufruf
Cls                                 ! Bildschirm löschen
Deffill ,2,8                        ! DEFFILL schwarz
Pbox 100,100,200,200                ! Schwarze Box zeichnen
Print "Ende = Maustaste gedrückt halten"
Repeat
  Pause 10
  Defmouse Ms1$                     ! Maus 1 (positiv)
  Pause 10
  Defmouse Ms2$                     ! Maus 1 (negativ)
  Pause 10
  Defmouse Ms3$                     ! Maus 2 (positiv)
  Pause 10
  Defmouse Ms4$                     ! Maus 2 (negativ)
  Pause 10
  Defmouse Ms5$                     ! Maus 3 (positiv)
  Pause 10
  Defmouse Ms6$                     ! Maus 3 (negativ)
  Pause 10
  Defmouse Ms7$                     ! Maus 4 (positiv)
  Pause 10
  Defmouse Ms8$                     ! Maus 4 (negativ)
Until Mousek
'
Procedure Dmouse(Mx%,My%,Mxa%,Mya%,Var Msp1$,Msp2$) ! V3.0-Kopf
  ' Procedure Dmouse(Mx%,My%,Mxa%,Mya%,Msp1%,Msp2%) ! V2.xx-Kopf
  '
```

```
' Liest einen 16*16-Pixel-Block als Mausform ein und
' liefert die Maus als Positiv und als Negativ.
' Für Hires/Midres/Lowres.
'
' Mx%  = Linke Quell-X-Koordinate
' My%  = Obere Quell-Y-Koordinate
' Mxa% = Aktionspunkt-X-Koordinate im Mausbild (0-15)
' Mya% = Aktionspunkt-Y-Koordinate im Mausbild (0-15)
' ' Msp1%= in V2.xx = Pointer ...          ----. welche nach
'          ... auf Rückgabe-String-Variable, ¦ Abschluß die
' Msp1$= in V3.0 Rückgabe-String-Variable, --' pos. Mausdaten
'                                              enthält.
' ' Msp2%= in V2.xx = Pointer ...          ----. welche nach
'          ... auf Rückgabe-String-Variable, ¦ Abschluß die
' Msp2$= in V3.0 Rückgabe-String-Variable, --' neg. Mausdaten
'                                              enthält.
Local Mspr$,Mcnt%,Mms$,Mspr1$,Mspr2%,Xb% ! Lokale Variablen
Xb%=Xbios(4)                          ! Auflösung holen
Boundary 0                            ! P-Rahmen aus (V3.0)
'  Dpoke Vdibase+34,0                 ! P-Rahmen aus (V2.xx)
Mms$=Mki$(Mxa%)+Mki$(Mya%)+Mki$(1)+Mki$(1)+Mki$(0)
'                                     ! Maus-String-Header
Get Mx%,My%,Mx%+15,My%+15,Mspr$ ! Bit-Muster speichern
Get Mx%-2,My%-2,Mx%+17,My%+17,Mspr1$ ! Hintergrund sichern
Deffill ,0,0                          ! Arbeitsbereich...
Pbox Mx%-1,My%-1,Mx%+16,My%+16 ! ...löschen
For I%=0 To 360 Step 45               ! einmal rundum
  Px%=Mx%+(Sin(I%*Pi/180)+0.5) ! X-Koordinate -. Bild in allen
  '                                   !               ¦ Richtungen je
  '                                   !               ¦ um ein Pixel
  Py%=My%+(Cos(I%*Pi/180)+0.5) ! Y-Koordinate -' versetzt,
  Put Px%,Py%,Mspr$,7                 ! im Transparentmodus zeichnen
Next I%
Get Mx%,My%,Mx%+15,My%+15,Mspr2$ ! Maske speichern
Put Mx%-2,My%-2,Mspr1$                ! Hintergrund restaurieren
Put Mx%,My%,Mspr$                     ! Mausbild restaurieren
'-----------------------------------------------------------.
For Mcnt%=0 To 15                     ! 16 Maskenzeilen in...   ¦
  Mms$=Mms$+Mki$(Dpeek(Varptr(Mspr2$)+6+Mcnt%*2*2^(2-Xb%)))!
  '                                   ! ...Maus-String einbinden ¦ M
Next Mcnt%                            ! nächste Maskenzeile     ¦ A
For Mcnt%=0 To 15                     ! 16 Mausbildzeilen in... ¦ U
  Mms$=Mms$+Mki$(Dpeek(Varptr(Mspr$)+6+Mcnt%*2))               ! S
  '                                   ! ...Maus-String einbinden ¦
Next Mcnt%                            ! nächste Bildzeile       ¦
'-----------------------------------------------------------'
'-----------------------------------------------------------.
' ** Zu diesem Block beachten Sie das Beispiel zu SPRITE **  ¦ S
' For Mcnt%=0 To 15                                          ! P
'   Mms$=Mms$+Mki$(Dpeek(Varptr(Mspr2$)+6+Mcnt%*2*2^(2-Xb%)))! R
'   Mms$=Mms$+Mki$(Dpeek(Varptr(Mspr$)+6+Mcnt%*2*2^(2-Xb%))) ! I
' Next Mcnt%                                                 ! T
'-----------------------------------------------------------' E
Boundary 1                            ! P-Rahmen ein (V3.0)
'  Dpoke Vdibase+34,1                 ! P-Rahmen ein (V2.xx)
Msp2$=Mms$                            ! Negativ-Maus-Rückgabe (V3.0)
'  *Msp2%=Mms$                        ! Negativ-Maus-Rückgabe (V2.xx)
Mid$(Mms$,7,4)=Mkl$(1)                ! Positiv-Flag im Maus-Header an
Msp1$=Mms$                            ! Positiv-Maus-Rückgabe (V3.0)
```

```
  ' *Msp1%=Mms$                         ! Positiv-Maus-Rückgabe (V2.xx)
Return
```

Unter ALERT finden Sie ein etwas ungewöhnliches Verfahren beschrieben, eigene Maus-Images zu installieren, die dann über DEFMOUSE 0 - 7 einsetzbar sind und resident im Speicher bleiben - auch wenn Sie das GFA-BASIC verlassen.

DEFTEXT { DEFT } Grafik-Text-Modi bestimmen

DEFTEXT [Farbe],[Art],[Winkel],[Größe],[Face]

Festlegung der Darstellung von Grafiktext. Außerhalb von GEM-Fenstern hat dieser Befehl nur Auswirkung auf die mit TEXT ausgegebenen Zeichen. Innerhalb von GEM-Fenstern können auch mit PRINT ausgegebene Zeichen beeinflußt werden.

Der innerhalb von ALERT- und FILESELECT-Boxen, sowie GEM-Fenstern verwendete Text kann zudem über VDI-Funktionen beeinflußt werden. Ebenso die darin verwendeten Linienarten. Diese lassen sich also bis zu einem gewissen Grad individuell verändern (siehe Beispiel 2).

Parameter Textfarbe siehe unter COLOR.

Art gibt die Textart an (5-Bit-Vektor):

```
Bit 0 gesetzt = bold        (fett)
Bit 1 gesetzt = lightend    (grau)
Bit 2 gesetzt = italic      (schräg/kursiv)
Bit 3 gesetzt = underlined  (unterstrichen)
Bit 4 gesetzt = outlined    (umrandet)
```

TEXTSTIL 0 TEXTSTIL 8 TEXTSTIL 16 TEXTSTIL 24
TEXTSTIL 1 TEXTSTIL 9 TEXTSTIL 17 TEXTSTIL 25
TEXTSTIL 2 TEXTSTIL 10 TEXTSTIL 18 TEXTSTIL 26
TEXTSTIL 3 TEXTSTIL 11 TEXTSTIL 19 TEXTSTIL 27
TEXTSTIL 4 TEXTSTIL 12 TEXTSTIL 20 TEXTSTIL 28
TEXTSTIL 5 TEXTSTIL 13 TEXTSTIL 21 TEXTSTIL 29
TEXTSTIL 6 TEXTSTIL 14 TEXTSTIL 22 TEXTSTIL 30
TEXTSTIL 7 TEXTSTIL 15 TEXTSTIL 23 TEXTSTIL 31

Aus den 32 möglichen Kombinationen dieser 5 Bits ergeben sich also 32 (0 - 31) verschiedene Textarten.

Winkel stellt den Textrotationswinkel ein:

```
   0 = Normale Lage.
 900 = Senkrecht von unten nach oben
1800 = Waagerecht von rechts nach links
2700 = Senkrecht von oben nach unten
```

Größe gibt die Höhe der Textzeichen (0 - 26) an, z.B.:

```
 4 = Icon-Schrift             6 = Farbschrift (8*8)
13 = Normalschrift (8*16)   26 = Mammut-Schrift
```

Face steht für den Index des gewünschten GDOS/VDI-Fonts. Siehe dazu GDOS?, VDIBASE+52, VDIBASE+68, VST_LOAD_FONTS() und VQT_NAME().

Beispiel 1 (nur Hires):

```
Graphmode 2
For J%=0 To 1
  For I%=0 To 15
    Deftext 1,I%+J%*16,,13       ! Textart einstellen
    Text 10+J%*80,20+I%*22,"TEXT  "+Str$(I%+J%*16)
  Next I%
Next J%
For J%=0 To 2700 Step 900
  Deftext 1,0,J%,13              ! Textwinkel einstellen
  Text 300,120," TEXT  "+Str$(J%)
Next J%
For J%=4 To 26
  Deftext 1,0,0,J%               ! Texthöhe einstellen
  Text 380,J%*17-60,"TEST-TEXT  "+Str$(J%)
Next J%
```

Beispiel 2 (VDI-Text):

```
Pbox 0,0,639,399
Ltype(1,2)                  ! Gestrichelte Linie
Ttype(1,4)                  ! Text-Typ "normal-kursiv"
Tsize(1,6)                  ! Text-Höhe 6
Fileselect "*.*","",A$      ! AES-FILESELECT
Ttype(1,1)                  ! Text-Typ fett
Tsize(1,9)                  ! Text-Höhe 9
Alert 2,"ALERT-Text",1,"BUTTON",A% ! AES-ALERT
Ttype(1,0)                  ! Text-Typ normal -. Standard
Tsize(1,13)                 ! Text-Höhe 13    -' Einstellung
Ltype(1,1)                  ! Standard-Voll-Linie
'
Procedure Ttype(Handle%,Type%) ! VDI-Textart
  Dpoke Contrl+2,0          ! CONTROL-Feld..
  Dpoke Contrl+6,1          ! ...vorbereiten
  Dpoke Contrl+12,Handle% ! VDI-Handle (siehe VDIBASE+40)
  Dpoke Intin,Type%         ! Textart (s.o.)
  Vdisys 106                ! Opcode
Return
Procedure Tsize(Handle%,Size%) ! VDI-Texthöhe
  Dpoke Contrl+2,0          ! CONTROL-Feld..
  Dpoke Contrl+6,1          ! ...vorbereiten
  Dpoke Contrl+12,Handle% ! VDI-Handle (siehe VDIBASE+40)
  Dpoke Intin,Size%         ! Texthöhe (s.o.)
  Vdisys 107                ! Opcode
Return
Procedure Ltype(Handle%,Lndef%) ! VDI-Linienmuster
  Dpoke Contrl+2,0          ! CONTROL-Feld..
  Dpoke Contrl+6,1          ! ...vorbereiten
  Dpoke Contrl+12,Handle% ! VDI-Handle (siehe VDIBASE+40)
  Dpoke Intin,7             ! Selbstdefiniertes Flag
  If Lndef%=>0 And Lndef%<7 ! Standard-Linien?
    Dpoke Intin,Lndef%      ! Dann Standard-Einstellung
    Vdisys 15               ! Defline-Opcode
  Else                      ! Frei definiertes Muster?
    Vdisys 15               ! Defline-Opcode
    Dpoke Intin,Lndef%      ! Eigenes Muster installieren
    Vdisys 113              ! User-Style-Opcode
  Endif
Return
```

GRAPHMODE { G } — Grafikmodus bestimmen

In V3.0: { GR }

GRAPHMODE Modus

Modus bestimmt den Operationsmodus, mit welchem Grafikausgaben in den bestehenden Hintergrund eingesetzt werden.

0, 1 = REPLACE

Das verwendete Grafik-Element (Pbox, Line etc.) wird vollflächig dargestellt. Alles, was sich darunter befindet, wird davon überdeckt und ersetzt.

```
Neuer Punkt = Farbmaske d. neuen Punktes AND neuer Punkt
```

2 = TRANSPARENT

Es werden nur dort Bildpunkte gesetzt, wo dem neuen Grafik-Element Farben zugeordnet wurden. Wo keine Farben darzustellen sind, bleibt der alte Hintergrund erhalten. Die neue Fläche erscheint also durchsichtig.

```
Neuer Punkt = (Farbmaske d. neuen Punktes AND neuer Punkt)
              OR (Farbe d. alten Punktes AND NOT neuer Punkt)
```

3 = XOR (EXCLUSIV ODER)

Ist in dem zu zeichnenden Grafik-Objekt ein Punkt gesetzt und dieser trifft auf einen Bildschirmpunkt gleicher Farbe, so wird der Punkt "gelöscht", d.h. er erscheint in der Hintergrundfarbe (Farbregister 0). Bei Aufeinandertreffen unterschiedlicher Farben ergibt sich die Farbe des neuen Punktes aus der XOR-Verknüpfung der beiden Punktfarben. Der bereits gezeichnete Hintergrundpunkt bezieht seine Farbe z.B. aus dem Farbregister 6 und der neu zu zeichnende Punkt aus dem Register 12:

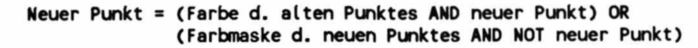
```
      0110      (dezimal 6)
XOR   1100      (dezimal 12)
----------------------------
  =   1010      (dezimal 10)
============================
```

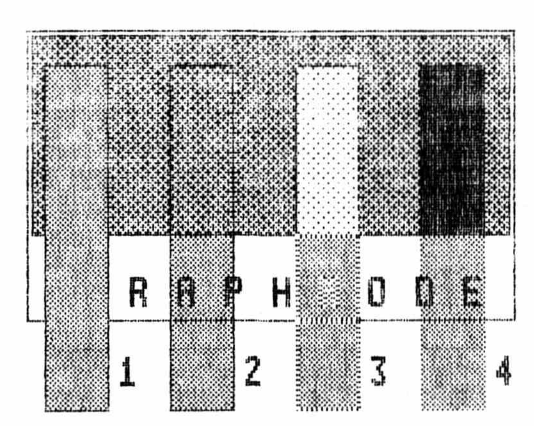
Der neue Punkt erhält seine Farbe aus Register 10. Setzt man zwei gleiche Grafik-Elemente in diesem Modus auf exakt dieselbe Stelle, wird das erste vom zweiten wieder gelöscht, ohne den vorherigen Hintergrund zu zerstören. Neuer Punkt = neuer Punkt XOR Farbe d. alten Punktes

4 = REVERS TRANSPARENT

Dieser Modus ist identisch mit Modus 2, nur daß die Farben revers (umgekehrt) dargestellt werden. Alles was im neuen Grafik-Element z.B. als weiß definiert wurde, wird schwarz dargestellt und alles was als schwarz definiert wurde, weiß. Wenn also zwei gleiche Grafik-

Elemente einmal im Modus 2 und einmal im Modus 4 dargestellt würden, so wäre das zweite das exakte Negativ des ersten.


Neuer Punkt = (Farbe d. alten Punktes AND neuer Punkt) OR
(Farbmaske d. neuen Punktes AND NOT neuer Punkt)



Beispiel:

```
Deffill ,2,5
Pbox 10,10,200,90
Box 8,8,202,124
Text 30,118,160,"GRAPHMODE"
Deffill ,2,4
For I%=1 To 4
  Graphmode I%
  Pbox I%*50-35,22,I%*50-20,160
  Text I%*50-20,80,I%
Next I%
```

SETCOLOR { SE } — Hardware-Farbregister einstellen

In V3.0: { SET }

SETCOLOR Reg,Rot,Grün,Blau
SETCOLOR Reg,Mischwert

Es kann die Farbe des mit Reg angegebenen Farbregisters (Hardware) entweder durch die Farbanteile Rot, Grün und Blau (jeweils 0 bis 7) oder durch einen Mischwert (1 bis 1911) definiert werden.

Mischwert = (Rotanteil*256)+(Grünanteil*16)+Blauanteil

Der Hardware-Registerindex ist nicht identisch mit dem Index der VDI-Register (vgl. COLOR).

Decodierungstabelle (Lowres):

SETCOLOR:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
COLOR :	0	2	3	6	4	7	5	8	9	10	11	14	12	15	13	1

COLOR :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SETCOLOR:	0	15	1	2	4	6	3	5	7	8	9	10	12	14	11	13

Decodierungstabelle (Midres):

SETCOLOR:	0	1	2	3
COLOR :	0	2	3	1

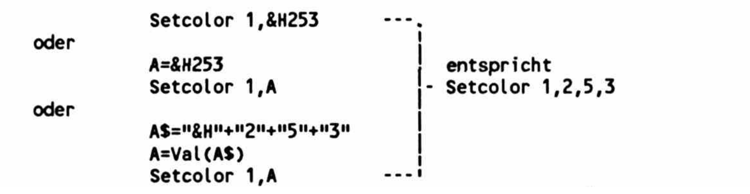
COLOR :	0	1	2	3
SETCOLOR:	0	3	1	2

Der Versuch, im Hires-Modus die Register zu verändern, wird nicht mit verschiedenen Grautönen belohnt. Schwarz (Register 1) bleibt schwarz und weiß (Register 0) bleibt weiß.

Bei verschiedenen Gelegenheiten ist es von Nutzen, die Registereinträge nicht mit drei unabhängigen Parametern vorzunehmen, sondern einen Gesamt-Mischungswert zu übergeben. GFA-BASIC unterscheidet, ob ein oder mehrere Farbparameter übergeben wurden. Wird nur der Parameter Mischwert übergeben, setzt sich die Farbmischung nach folgendem Muster zusammen:

Soll z.B. ein Blauanteil von 3, ein Rotanteil von 2 und ein Grünanteil von 5 übergeben werden, errechnet sich der Wert, indem man den Rotanteil mit 256 (2^8) multipliziert, den Grünanteil mit 16 (2^4) und den Blauanteil mit 1 (2^0). In diesem Beispiel müßte der Wert 595 übergeben werden, um die gewünschte Farbmischung zu erhalten.

Durch die Möglichkeit, mit Hexadezimalzahlen zu arbeiten, kann man diese Berechnung folgendermaßen vereinfachen:



```
          Setcolor 1,&H253          ---.
oder                                   |
          A=&H253                      |  entspricht
          Setcolor 1,A                 |- Setcolor 1,2,5,3
oder                                   |
          A$="&H"+"2"+"5"+"3"          |
          A=Val(A$)                    |
          Setcolor 1,A              ---'
```

So lassen sich 512 verschiedene Farben (3 Farben mit je 8 Abstufungen = 8*8*8 = 512 Farben) erzeugen.

Die Bits 0 - 3 von Mischwert bestimmen dabei den Rotanteil, die Bits 4 - 7 den Grünanteil und die Bits 8 - 11 den Blauanteil der Farbe.

Drei Register haben eine besondere Bedeutung:

```
Register 0  =  Hintergrundfarbe
Register 1  =  PRINT-Ausgabetextfarbe
Register 3  =  Textfarbe des BASIC-Editors
```

Leider ist der Goethesche Farbkreis hier nicht anwendbar, da hier nicht die drei Elementarfarben Rot, Blau und Gelb gemischt werden. Um sich bei der Farbwahl zu orientieren, müßte aus Grün die Farbe Blau subtrahiert werden, um Gelb zu erhalten Trotzdem kann man sich seiner als allgemeine Richtlinie bedienen:

```
            ROT
VIOLETT          ORANGE
          WEISS
BLAU                GELB
           GRÜN
```

Zur Farbphysik:

Es existieren drei Elementarfarben : ROT, BLAU, GELB.

Die 1:1-Mischung zweier dieser Farben ergibt die Komplementärfarbe (Gegenfarbe) der nicht beteiligten dritten Farbe.

1:1 Mischung von ROT und BLAU ergibt VIOLETT (= Komplement zu GELB)

1:1 Mischung von ROT und GELB ergibt ORANGE (= Komplement zu BLAU)

1:1 Mischung von BLAU und GELB ergibt GRÜN (= Komplement zu ROT)

1:1:1 Mischung von ROT, BLAU und GELB ergibt WEISS.

Ist keine der Elementarfarben beteiligt, ergibt sich die "Nichtfarbe" SCHWARZ.

Beispiel (nur für Lowres):

```
Dim C%(15)                  ! DIM Farb-Speicher
Setcolor 0,0                ! Hintergrund schwarz
For I%=1 To 15              ! 15 Register (außer 0) >-------.
  If I%<8                   ! 7 Werte von weiß zu grün      |
    C%(I%)=I%*16            ! Grünanteil erhöhen            |
  Else                      ! 8 Werte von gelb zu rot       |
    C%(I%)=7*256+(15-I%)*16 ! Grünanteil vermindern         |
  Endif                                                     |
  Setcolor I%,C%(I%)        ! Farbe setzen                  |
Next I%                     ! Nächstes Register <-----------'
For J%=0 To 4               ! 5 mal >------------------------.
  Restore                   ! Data-Zeiger setzen             |
  For I%=1 To 15            ! 15 Farben  >----------------.  |
    Read A%                 ! COLOR-Register holen        |  |
    Color A%                ! BOX- und CIRCLE-Farbe setzen|  |
    Box 60+J%*20+I%,J%*20+I%,260-J%*20-I%,199-J%*20-I%    |  |
    Circle 160,100,J%*15+I%*10                            |  |
    Circle 160,100,J%*15+I%*10+5                          |  |
  Next I%                   ! Nächste Farbe <-------------'  |
Next J%                     ! Nächster Offset <--------------'
Do                          ! Endlos-Schleife >-------------.
  B%=C%(15)                 !--.                            |
  For I%=14 Downto 1           !                            |
    C%(I%+1)=C%(I%)            !- Farben rotieren           |
  Next I%                      !                            |
  C%(1)=B%                  !--'                            |
  For I%=1 To 15            ! 15 Register >------------.    |
    For J%=1 To 100         !    Kleine...             |    |
    Next J%                 !    ...Pause              |    |
    Setcolor I%,C%(I%)      ! Neue Farbe  setzen       |    |
  Next I%                   ! Nächstes Register <------'    |
Loop                        ! <-----------------------------'
Data 2,3,6,4,7,5,8,9,10,11,14,12,15,13,1, Umrechnungstabelle
```

Dieses Beispiel produziert eine Farbspielerei, die für Interessierte leicht zur Meditationshilfe (für gestreßte Programmierer) oder zur Bio-Feedback-Methode "zweckentfremdet" werden kann. Die Farben pulsieren mit einer Frequenz von ca. 58 Zyklen pro Minute (ca. optimale Pulsfrequenz bei Entspannung). Die Frequenz läßt sich leicht mit der Pausen-Schleife im letzten Block verändern.

Wenn Sie übrigens das Flimmern Ihres Farbmonitors stört, können Sie dies unterdrücken, indem Sie SPOKE &HFF820A,0 eingeben (60 Hz). Durch SPOKE &HFF82 wird die Normaleinstellung restauriert.

Version 3.0

VSETCOLOR { VSE } VDI-Farbregister einstellen

VSETCOLOR Reg,Rot,Grün,Blau
VSETCOLOR Reg,Mischwert

Dieser Befehl ist in seiner Syntax absolut identisch mit SETCOLOR. Der einzige Unterschied ist, daß hiermit in Version V3.0 die Farb-Register direkt mit dem VDI-Index angesprochen werden können.

Wenn Sie mit VSETCOLOR z.B. das Farbregister 7 verändert haben und anschließend durch COLOR, DEFTEXT, DEFFILL etc. eine Farbe bestimmen, können Sie davon ausgehen, daß Sie auch die Farbe verwenden, die Sie vorher durch VSETCOLOR angegeben haben.

Für VSETCOLOR gelten dieselben Decodierungs-Tabellen, die unter SETCOLOR und COLOR angegeben wurden. Für COLOR in den Tabellen können Sie also ebensogut VSETCOLOR einsetzen.

Wie üblich folgt nun eine kleine Routine für die V2.xx-Versionen. Der Aufruf ist fast mit dem VSETCOLOR-/SETCOLOR-Befehl identisch. Es werden allerdings generell 4 Parameter erwartet. Der erste ist das gewünschte Register als VDI-Index. Wollen Sie - wie bei (V)SETCOLOR - einen Mischwert angeben, so müssen Sie die beiden Parameter G% und B% als Minuswert (-1) angeben. In diesem Fall wird der Parameter R% als Mischwert interpretiert.Ist G% oder B% gleich oder größer Null, so werden die drei letzten Parameter als die RGB-Farbanteile gewertet.

```
@Vsetcolor(5,4,1,7)          ! RGB-Anteile übergeben
@Vsetcolor(5,1047,-1,-1)     ! Mischwert übergeben
'
Procedure Vsetcolor(Reg%,R%,G%,B%)
  Local Tabelle$
  Tabelle$=Chr$(0)+Chr$(15)+Chr$(1)+Chr$(2)+Chr$(4)+Chr$(6)
  Tabelle$=Tabelle$+Chr$(3)+Chr$(5)+Chr$(7)+Chr$(8)+Chr$(9)
  Tabelle$=Tabelle$+Chr$(10)+Chr$(12)+Chr$(14)+Chr$(11)+Chr$(13)
  If G%=>0 And B%=>0
    Setcolor Asc(Mid$(Tabelle$,Reg%+1,1)),R%
  Else
    Setcolor Asc(Mid$(Tabelle$,Reg%+1,1)),R%,G%,B%
  Endif
Return
```

9.2 Objektgrafikbefehle

BOX, PBOX { B, PB } **Rechteck zeichnen**

In V3.0: { BO, PB }

[P]BOX X_links,Y_oben,X_rechts,Y_unten

X_links/Y_oben und X_rechts/Y_unten bezeichnen die diagonal gegenüber liegenden Ecken eines Rechtecks, das entweder als Linienzug (BOX) oder mit dem aktuellen DEFFILL-Füllmuster (PBOX) gezeichnet wird.

Beispiel: Wenn es darum geht, ein größeres Raster mit PBOX zu füllen (z.B. in einer Lupe), sollte man - wenn irgend möglich - versuchen, die PBOX durch eine PUT-Fläche zu ersetzen. Das geht erheblich schneller. Bei kleinen Rastern fällt der Geschwindigkeitsunterschied nicht besonders auf, bei größeren Rastern ist es dagegen schon ein merklicher Unterschied, ob das Raster in 2 oder in 4 Sekunden gefüllt wird. Um das zu demonstrieren, folgt eine "kleine" Routine, die es ermöglicht, einen beliebigen Bildausschnitt zu vergrößern und/oder ihn für eine spätere Verwendung (raster-indiziert) in einem Feld zwischenzuspeichern.

Hier ist eine kleine Bedienungsanleitung nötig. Sie können den Bildbereich, in dem der vergrößerte Ausschnitt dargestellt werden soll, beliebig bestimmen. Außerdem kann die Rastergröße und die Breite und Höhe eines vergrößerten Rasterpunktes angegeben werden. Es ist ungünstig, wenn sich der Wiedergabe-Ausschnitt mit dem zu vergrößernden Ausschnitt überschneidet. Der Wiedergabe-Ausschnitt wird - sollten seine Eck-Koordinaten außerhalb des Bildschirms liegen - von der Routine auf den sichtbaren Bildschirm begrenzt.

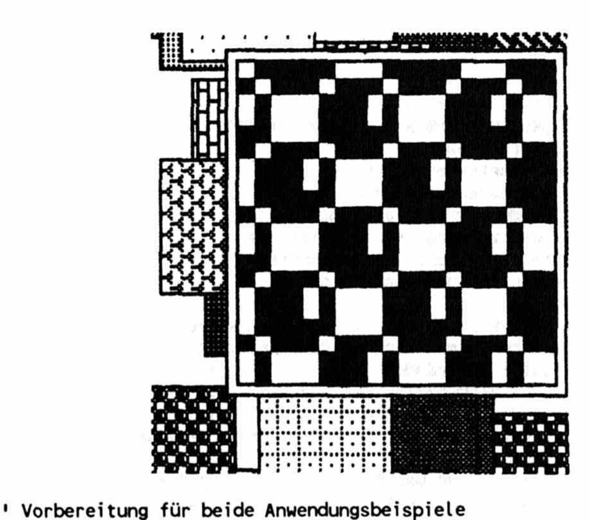
Ist die Routine aufgerufen, so wird ggfs. der vergrößerte Ausschnitt gezeichnet und die Routine wartet dann darauf, daß mit der rechten Maustaste der Lupeninhalt verändert wird. Verlassen wird die Lupe zu jedem beliebigen Zeitpunkt mit Druck auf die rechte Maustaste. Daran anschließend wird der Lupenhintergrund selbstständig restauriert.

Der Lupeninhalt wird verändert, indem mit der linken Maustaste der gewünschte Lupenpunkt angeklickt wird. Der Punkt verändert seine Farbe. Diese Farbe behält er auch bei Mausbewegungen solange bei, bis die Maustaste wieder losgelassen wird. In Midres und Lowres wird

bei jedem Mausklick auf einen Lupenpunkt sein Farbindex um 1 vermindert. Klicken Sie also z.B. in Lowres 16 mal auf denselben Punkt, nimmt er nacheinander alle 16 möglichen Farben an. In Farbe ist es ratsam, bei einem Punkt-Klick die Maustaste gedrückt zu halten, um zu sehen, welche Farbe sich ergibt. Solange Sie die Taste gedrückt halten, können Sie mit dieser Farbe innerhalb der Lupe zeichnen. Wird die Taste losgelassen, wird **der** Farbindex um 1 vermindert, auf dem sich die Maus zum Zeitpunkt des Klicks befindet. Haben Sie als Flag 1 angegeben, so wird gleichzeitig auch der originale Bildschirmbereich entsprechend der Lupenänderung verändert. Nach Rückkehr aus der Prozedur ist der aktuelle Lupeninhalt in dem Feld Larr%(Rasterbreite,Rasterhöhe) gespeichert und kann damit auch außerhalb der Prozedur verwendet werden.

Soll die Lupe nicht dargestellt werden, geben Sie als Flag 2 an. In diesem Fall wird nur das Feld Larr%() mit dem angegebenen Bildausschnitt gefüllt. Der Feldindex entspricht der Lage der Punkte im Raster. Die Indizierung beginnt mit Null. Z.B. Larr%(0,0) enthält den Farbwert des 1. Punktes links oben in der Ausschnittecke und z.B. Larr%(12,6) den Farbwert des Rasterpunktes mit den Rasterkoordinaten 13. Punkt von links/7. Punkt von oben. Grundsätzlich ist zu bedenken, daß das Feld - abhängig von der Rastergröße - eine ganze Menge Speicherplatz verbrauchen kann. Ein Raster von z.B. 50*50 Punkten benötigt (50*50*4) 10000 Bytes.

Weitere Informationen finden Sie unten in der Beschreibung der Prozedur. Die Prozedur ist - wie aus der Beschreibung erkennbar - in Hires, Midres und Lowres lauffähig, sowie in allen GFA-BASIC-Versionen ab V2.0.



```
' Vorbereitung für beide Anwendungsbeispiele
'-------------------------------------------
Xt%=2-Sgn(Xbios(4))              ! X-Auflösungsteiler
Yt%=Min(2,3-Xbios(4))            ! Y-Auflösungsteiler
For I%=0 To 100                  ! 100 mal
  Deffill Random(Max(1,(2^(2-Xbios(4)))^2-1))+1,2,Random(22)
  X%=Random(590/Xt%)             ! PBOX mit zufälliger...
  Y%=Random(350/Yt%)             ! ...Farbe, Muster und...
  Pbox X%,Y%,X%+50/Xt%,Y%+50/Yt% ! ...Position zeichnen
Next I%
'
' Anwendungsbeispiel 1:
' --------------------
Print "Lupe mit rechter Maustaste aufrufen"
Do                               ! Endlos-Schleife
  Graphmode 3                    ! XOR-Modus für Flimmerbox
  Mouse Xx%,Yy%,K%               ! Maus-Status holen
  Xx%=Max(130,Xx%)               ! Box-X-Position begrenzen
  Yy%=Max(130,Yy%)               ! Box-Y-Position begrenzen
  Box Xx%-1,Yy%-1,Xx%+20,Yy%+20  ! Flimmerbox...
  Box Xx%-1,Yy%-1,Xx%+20,Yy%+20  ! ...zeichnen
  If Mousek=2                    ! Rechte Maustaste gedrückt?
    Graphmode 1                  ! REPLACE-Modus an
    Lupe(Xx%-20*6,Yy%-20*6,20,20,6,6,1) ! Aufruf
    Hidem                        ! Maus aus
  Endif
Loop
'
' Anwendungsbeispiel 2 (vorher Beispiel 1 löschen):
' ------------------------------------------------
Print "Maushintergrund (20*20) in Feld Larr%() einlesen.
Print "beliebige Maustaste drücken"
Repeat                           ! Auf...
Until Mousek                     ! Mausklick warten
@Lupe(Mousex-20*6,Mousey-20*6,20,20,6,6,2) ! Lupe aufrufen
Cls                              ! Bildschirm löschen
For I%=0 To 20-1                 ! 20 Zeilen
  For J%=0 To 20-1               ! 20 Spalten
    Color Larr%(I%,J%)           ! Punktfarbe setzen
```

```
    Plot 100+I%,100+J%         ! Punkt zeichnen
  Next J%
Next I%
'
Procedure Lupe(Xp%,Yp%,Br%,Ho%,Sx%,Sy%,Flg%)
  ' Für Hires/Midres/Lowres
  ' Xp% = X-Koordinate der Lupenbox
  ' Yp% = Y-Koordinate der Lupenbox
  ' Br% = Breite des zu vergrößernden Rasters
  ' Ho% = Höhe des zu vergrößernden Rasters
  ' Sx% = Breite eines Lupen-Rasterpunktes
  ' Sy% = Höhe eines Lupen-Rasterpunktes
  ' Flg%= Flag
  '        0 = Lupe wird gezeichnet und kann verändert
  '            werden. Der Original-Ausschnitt bleibt
  '            dabei unverändert.
  '        1 = Lupe wird gezeichnet und kann verändert
  '            werden. Der Original-Ausschnitt wird
  '            entsprechend der Lupenänderungen ebenfalls
  '            verändert.
  '        2 = Es wird nur das Rasterfeld mit den Punktfarben
  '            des Original-Rasters gefüllt. Die Lupe wird
  '            nicht gezeichnet.
  ' Das Rasterfeld Larr%() wird auch bei Modus 0 und 1 gefüllt
  ' und kann nach Rückkehr aus der Routine ausgewertet werden.
  '
  Local I%,J%,Lb$,Mx%,My%,Mk%,Xp2%,Yp2%,Pkt%
  Local Xt%,Yt%,Pnt%,Mxx%,Myy%,Pxr%,Pyr%
  Xt%=2-Sgn(Xbios(4))                   ! X-Auflösungsteiler
  Yt%=Min(2,3-Xbios(4))                 ! Y-Auflösungsteiler
  Sx%=Min(Sx%,16)                       ! Max. Punktbreite = 16
  Xp%=Min(639/Xt%,Max(Xp%-10,0))        ! Koordinaten der...
  Yp%=Min(399/Yt%,Max(Yp%-10,0))        ! ...Wiedergabe-Box...
  Xp2%=Min(639/Xt%,Max(Xp%+Br%*Sx%+9,0)) ! ...auf Bildschirm...
  Yp2%=Min(399/Yt%,Max(Yp%+Ho%*Sy%+9,0)) ! ...begrenzen
  Get Xp%,Yp%,Xp2%,Yp2%,Lb$             ! Hintergrund sichern
  Erase Larr%()                         ! Rasterfeld löschen
  Dim Bl$((2^(2-Xbios(4)))^2),Larr%(Br%,Ho%) ! DIM PUT-
  '                                     ! und Rasterfeld
  If Flg%<>2                            ! Lupe zeichnen?
    For I%=0 To (2^(2-Xbios(4)))^2      ! Alle möglichen Farben
      ' (2^(2-Xbios(4)))^2 ergibt:
      '        in Hires  =   1
      '        in Midres =   4
      '        in Lowres =  16
      Deffill I%,2,8                    ! Füllmusterfarbe setzen
      Pxr%=Xp%+4+Min(Br%*Sx%,18)        ! Breite u. Höhe der...
      Pyr%=Yp%+4+Min(Ho%*Sy%,18)        ! ...PBOX ermitteln
      Pbox Xp%+4,Yp%+4,pxr%,pyr%        ! PBOX vollfarbig zeichnen
      Get Xp%+5,Yp%+5,Xp%+5+15,Yp%+5+15,Bl$(I%) ! PUT-Box...
      '                                 ! ..speichern (PBOX-Ersatz)
      Mid$(Bl$(I%),1,4)=Mki$(Max(1,Sx%-1))+Mki$(Max(1,Sy%-1))
      ' PUT-Header an gewünschte Punktgröße anpassen
    Next I%
    Deffill 0,0,0                       ! DEFFILL weiß
    Color 1                             ! COLOR für Rahmenbox
    Pbox Xp%,Yp%,Xp2%,Yp2%              ! Lupenhintergrund löschen
    Box Xp%,Yp%,Xp2%,Yp2%               ! Lupenrahmen zeichnen
    Box Xp%+4,Yp%+4,Xp2%-4,Yp2%-4       ! Lupenrahmen zeichnen
```

```
      Endif
      Mouse Mx%,My%,Mk%                    ! Aktuelle Mauskoordinate
      For I%=0 To Br%-1                    ! Raster-X-Index
        For J%=0 To Ho%-1                  ! Raster-Y-Index
          Pnt%=Point(Mx%+I%,My%+J%)        ! Punktfarbe ermitteln
          If Pnt%                          ! Punkt gesetzt?
            Larr%(I%,J%)=Pnt%              ! Punktfarbe in Feld schreiben
            If Flg%<>2                     ! Lupe soll gezeichnet werden?
              Put Xp%+5+I%*Sx%,Yp%+5+J%*Sy%,Bl$(Pnt%) ! Lupenpunkt...
            Endif                                     ! ...zeichnen
          Endif
        Next J%
      Next I%
      If Flg%<>2                           ! Lupe ist gezeichnet?
        Repeat                             ! Lupen-Schleife
          Mouse Mxx%,Myy%,Mk%              ! Maus-Status holen
          Showm                            ! Maus an
          Repeat                           ! warten...
          Until Mousex<>Mxx% Or Mousey<>Myy% Or Mousek
          '                                ! ...auf Mausaktion
          If Mxx%>Xp%+5 And Mxx%<Xp%+4+Br%*Sx%   ! X-Maus in der Lupe?
            If Myy%>Yp%+5 And Myy%<Yp%+4+Ho%*Sy% ! Y-Maus in der Lupe?
              Mxx%=Int((Mxx%-(Xp%+5))/Sx%) ! Koordinaten auf...
              Myy%=Int((Myy%-(Yp%+5))/Sy%) ! ...Rasterindex umrechnen
              If Mk%=1                     ! Linke Maustaste gedrückt?
                If Flg%=1                  ! Originalpunkt ändern?
                  Plot Mx%+Mxx%,My%+Myy% ! Dann Punkt setzen
                Endif
                Larr%(Mxx%,Myy%)=Pkt%      ! Rasterfeld aktualisieren
                Put Xp%+5+Mxx%*Sx%,Yp%+5+Myy%*Sy%,Bl$(Pkt%)
                '                          ! Lupenpunkt setzen
              Else                         ! linke Maustaste ist aus!
                Pkt%=Point(Xp%+5+Mxx%*Sx%,Yp%+5+Myy%*Sy%)-1
                '                          ! Punktfarbe um 1 vermindern
                If Pkt%<0                  ! Farbindex < 0?
                  Pkt%=Max(1,(2^(2-Xbios(4)))^2-1) ! Index auf Maximum
                Endif
                Color Pkt%                 ! Originalpunktfarbe setzen
              Endif
            Endif
          Endif
        Until Mk%=2                        ! Rechte Maustaste gedrückt?
        Pause 5                            ! Kleine Klick-Pause
      Endif
      Erase Bl$()                          ! PUT-Box-Feld löschen
      Put Xp%,Yp%,Lb$                      ! Hintergrund restaurieren
    Return
```

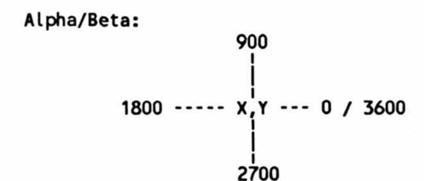
CIRCLE, PCIRCLE { C, PC } Kreis(bogen) zeichnen

In V3.0: { CI, PC }

[P]CIRCLE X_cent,Y_cent,Radius [,Alpha,Beta]

"X_cent/Y_cent" bestimmen den Kreismittelpunkt. "Radius" ist der halbe Kreisdurchmesser. Wird die Option "Alpha/Beta" (in 10tel Grad

-> 0 bis 3600) genutzt, so wird ein Kreisbogen mit dem Startwinkel Alpha und dem Endwinkel Beta gezeichnet.



ELLIPSE, PELLIPSE { ELL, PE } Ellipse(nbogen) zeichnen

[P]ELLIPSE Xcent,Ycent,Xrad,Yrad [,Alpha,Beta]

"Xrad" ist der Ellipsenradius in X-Richtung und "Yrad" der in Y-Richtung. "Xcent/Ycent" legt den Ellipsenmittelpunkt fest. Zur Option "Alpha/Beta" siehe Ausführungen bei CIRCLE.

FILL { FI } Flächen mit Muster füllen

FILL Xpos,Ypos
FILL Xpos,Ypos [,Farbe] **(nur V3.0)**

"Xpos/Ypos" gibt die Lage des Bildschirmpunktes an, von dem aus sich der Füllvorgang ausbreitet, bis er an geschlossene Grenzen stößt.

FILL ist - **nur** in V2.xx - ggfs. mit Vorsicht zu genießen. Dies hat zwei Gründe. Der eine ist, daß das ausbreitende Füllmuster die kleinsten (1 Pixel) Schlupfwinkel entdeckt, durch die es sich weiter ausbreiten kann. Das führt ggfs. dazu, daß Bereiche, die nicht gefüllt werden sollen, durch den FILL-Befehl zunichte gemacht werden und es dann oft unmöglich ist, diesen Bereich wieder vom unerwünschten Füllmuster zu säubern.

Die zweite recht unangenehme Eigenschaft in V2.xx ist, daß bei größeren Füllflächen, die schon mit einem nicht geschlossenen Füllmuster ausgefüllt sind, je nach Größe der Füllfläche sehr, sehr viel Zeit vergehen kann, bis die Arbeit vollendet ist. In dieser Zeit geschieht außer dem Füllen **nichts**. FILL wird ausschließlich vom GEM gesteuert und ist dann auch nicht mehr durch irgendwelche Abbruch-Funktionen zu unterbrechen.

Sollte Ihnen in V2.xx das Malheur passieren, ein offenes Füllmuster mit **sich selbst** auf großen Flächen zu füllen, kann der Eindruck ent-

stehen, daß Ihnen der Computer Adieu gesagt hat. Lassen Sie ihm Zeit, er wird sich dann schon wieder melden. Also, nicht gleich zum Reset-Knopf greifen! Der Grund für dieses Schneckentempo ist darin zu suchen, daß im Farb-Betrieb der Computer aufgrund der 16 verschiedenen Farben, die dann gleichzeitig darstellbar sind, eine gewaltige Rechenleistung zu vollbringen hat. Im Monochrom-Modus macht er da keine Ausnahme. Obwohl dann nur zwei Farben vorhanden sein können, werden trotzdem alle 16 möglichen Farbregister in die Kontrolle einbezogen.

In Version V3.0 kann der Füllvorgang durch die GFA-Break-Funktion abgebrochen werden. Die oben beschriebenen V2.xx-Probleme lassen sich also - Ostrowski macht es möglich - durch die Tastenkombination <Control><Shift><Alternate> weitestgehend unterbinden. Die Abbruch-Tasten müssen dabei gedrückt gehalten werden, da intern erst die aktuelle Füll-Sequenz beendet wird.

Außerdem kann in Version V3.0 optional mit dem Parameter "Farbe" ein Farbwert angegeben werden. Es werden dann ausschließlich Bildschirmpunkte mit der angegebenen Farbe als Füllbegrenzung gewertet. Alle anderen Punkte (und Farben) werden gefüllt. Hat z.B. der Bildpunkt Xpos/Ypos die angegebene Farbe, wird der Füllvorgang sofort abgebrochen.

Bei eingeschaltetem Clipping (siehe CLIP) wird generell nur bis an die Grenzen des CLIP-Ausschnitts gefüllt.

Beispiel (nur Lowres):

```
Deffill 7,2,6
Pbox 10,10,310,190
Deffill 11,2,8
Pbox 90,40,180,90
For I%=1 To 15
  Deffill I%,2,8
  Pcircle Random(80)+85,Random(50)+45,10
Next I%
Deffill 4,2,8
Fill 176,44,7          ! V3.0-Aufruf
' Fill 176,44          ! V2.xx-Aufruf
```

POLYFILL { POLYF } Polygon zeichnen, gefüllt

POLYFILL Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]

Es gelten die gleichen Ausführungen wie zu POLYLINE. Nur daß zusätzlich die zwischen den einzelnen Linien liegenden Flächen mit dem eingestellten Füllmuster ausgefüllt werden.

Ein Beispiel finden Sie unter POLYLINE.

POLYMARK { POLYM } Polygon-Eckpunkte markieren

POLYMARK Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]

Es gelten die gleichen Ausführungen wie zu POLYLINE. Nur daß anstatt eines Linienzuges das eingestellte Markierungssymbol auf die definierten Eckpunkte gezeichnet wird.

Ein Beispiel finden Sie unter POLYLINE.

RBOX, PRBOX { RB, PRB } Rechteck abgerundet zeichnen

[P]RBOX X_links,Y_oben,X_rechts,Y_unten

Es gelten die gleichen Ausführungen wie zu BOX/PBOX. Das Rechteck wird jedoch mit abgerundeten Ecken gezeichnet.

TEXT { T } Text im Grafikmodus ausgeben

TEXT Xt,Yt [,[-]Länge],"Text"

"Text" kann sowohl direkt als Text, als String-Variable, oder als zusammengesetzter Textausdruck (A$+Str$("1")) angegeben werden. "Xt,Yt" steht für den Bildschirmpunkt, an den der Text linksbündig angelegt wird.

Die Ausgaben beschränken sich auf den Bildschirm, bzw. auf ein mit OPNVWK() geöffnetes Ausgabegerät (siehe dort).

Der zur Ausrichtung des Textes ausschlaggebende Punkt ist immer der, der bei normaler Lage des Textes (waagerecht von links nach rechts) in der Zeichenbox des ersten Zeichens (auch bei Leerzeichen)

links unten liegt. Mit diesem Punkt wird der Text generell an die angegebene X/Y-Position angelegt.

Die Option Länge bewirkt, daß der Text in seiner optischen Länge der Vorgabe angepaßt wird. Dies geschieht durch Streckung oder Stauchung der Buchstabenabstände (wenn Länge positiv) oder der Wortabstände (wenn Länge negativ).

Beispiel:

```
Deffill ,2,1
Pbox 0,0,639,399
Deftext 1,16,0,16
For I%=1 To 4
  Graphmode I%
  Tx$="TEXT (normal) GRAPHMODE "+Str$(I%)
  Text 10,20+I%*30,0,Tx$
  Tx$="TEXT (Zeichendehng.) GRAPHMODE "+Str$(I%)
  Text 10,140+I%*30,600,Tx$
  Tx$="TEXT (Wortdehnung) GRAPHMODE "+Str$(I%)
  Text 10,260+I%*30,-600,Tx$
Next I%
```

9.3 Strich-/Punktgrafik

DRAW { DR } — Punkte zeichnen und verbinden

DRAW TO Xpos,Ypos
DRAW X1,Y1 [TO X2,Y2 [TO X3,Y3...]]

Die erste Syntax-Variante verbindet den durch Xpos/Ypos bezeichneten Punkt mit dem zuletzt durch DRAW, PLOT oder LINE gezeichneten Grafik-Punkt. Die zweite Variante zeichnet einen beliebig langen Linienzug durch die angegebene Punkte-Kette. Wird die optionale Punkte-Kette weggelassen (DRAW X1,Y1) wird an der angegebenen Position ein einzelner Punkt gesetzt (vgl. PLOT).

Beispiel: Damit Sie sich nicht soviel Arbeit machen müssen, die einzelnen Punktkoordinaten einer Kette einzutippen, folgt nun ein kleines Hilfsprogramm. Wenn Sie es gestartet haben, können Sie mit einem Druck auf die linke Maustaste einzelne Punkte zeichnen. Es sind zwei Arrays dimensioniert, die die jeweilige X- und Y-Koordinate des gerade gezeichneten Punktes festhalten. Jedes dieser beiden Arrays hat 1000 Elemente, d.h., daß Sie maximal 1000 Punkte zeichnen können. Wenn Sie diese Punkteanzahl erreicht haben oder die rechte Maustaste

drücken, werden die beiden Arrays in Form von DATA-Zeilen unter dem Namen DRAW.LST auf Diskette abgespeichert.

Anschließend wird das erste Programm beendet. Laden Sie nun mit Merge die geschriebene DATA-Datei in den Arbeitsspeicher und starten Sie das zweite Programm. Sie werden sehen, daß Ihre Zeichnung nun anhand der DATA-Zeilen und des DRAW-Befehls neu auf den Bildschirm gebracht wird.

Programm 1:

```
Dim Px(1000),Py(1000)         ! DIM Punkte-Speicher
Do                            ! Eingabe-Schleife
  Mouse X,Y,K                 ! Maus-Status holen
  If K=1                      ! Linke Maustaste gedrückt?
    Pause 1                   ! Kleine Klickpause
    Draw X,Y                  ! Punkt zeichnen
    Inc Count                 ! Punktezähler +1
    Px(Count)=X               ! X-Koordinate speichern
    Py(Count)=Y               ! Y-Koordinate speichern
  Endif
  Exit If K=2 Or Count=1000 ! 1000 Punkte erreicht oder...
Loop                          ! ...rechte Maustaste gedrückt?
Open "O",#1,"DRAW.LST"        ! Datei öffnen
Print #1;"D.rawkoos:";Chr$(13) ! DATA-Label schreiben
For J=1 To 100                ! 100 DATA-Zeilen
  Print #1;"D ";              ! DATA schreiben
  For I=1 To 10               ! 10 Koordinatenpaare je Zeile
    Inc Ci                    ! Indexzähler +1
    Print #1;Str$(Px(Ci));",";Str$(Py(Ci)); ! X/Y schreiben
    Exit If Ci=> Count        ! Abbruch, wenn Anzahl < 1000
    If I<10                   ! Zeilenende noch nicht erreicht?
      Print #1;",";           ! Dann Komma schreiben
    Endif
  Next I                      ! Nächstes Koordinatenpaar
  Print #1;Chr$(13)           ! CR-Zeilenende schreiben
  Exit If Ci=> Count          ! Abbruch, wenn Anzahl < 1000
Next J                        ! Nächste DATA-Zeile
Print #1;"D ";Str$(1111)      ! DATA-Endmarke schreiben
Close #1                      ! Datei schließen
Edit                          ! Programmende
```

Programm 2:

```
Dim Px(1000),Py(1000)         ! DIM Punkte-Speicher
Restore D.rawkoos             ! DATA-Zeiger setzen
Read Px(1),Py(1)              ! 1. Koordinatenpaar lesen
Plot Px(1),Py(1)              ! und zeichnen
For I=2 To 1000               ! Restliche Koordinaten
  Read Px(I),Py(I)            ! Lesen
  ' Pbox Px(I),Py(I),Px(I)+10,Py(I)+10
  ' Ersetzen Sie die Zeile mit dem Draw-Befehl durch diese
  ' Pbox-Zeile und schon haben Sie einen neuen Effekt.
  Draw  To Px(I),Py(I)        ! Und zeichnen
```

```
    Exit If Px(I)=1111          ! Abbruch, wenn Ende erreicht
  Next I                        ! Nächstes Paar
```

Version 3.0

DRAW $ {DR} Plotter-(Turtle-)Grafik

DRAW Def$[,Const[,"Def"[,Var[,...]]]]

Erlaubt eine Plotter-Simulation auf dem Bildschirm (LOGO-Turtle-Grafik). In Def$/"Def" können wahlweise als alphanumerischer Ausdruck, als String-Variable oder Textkonstante Turtle-Kommandos angegeben werden. Als Kommandos sind folgende Kürzel vorgesehen:

(n enthält den jeweils anzugebenden Wert, x,y die betreffenden Koordinaten)

fd n	(forward) Bewege Stift n Pixel vorwärts.
bk n	(backward) Bewege Stift n Pixel rückwärts.
sx n	Skalierung aller bei fd und bk angegeben Werte in X-Richtung mit dem Wert n.
sy n	Skalierung aller bei fd und bk angegeben Werte in Y-Richtung mit dem Wert n.
sx 0	X-Skalierung ausschalten.
sy 0	Y-Skalierung ausschalten.
lt n	(left turn) drehe Stift um n Grad nach links.
rt n	(right turn) drehe Stift um n Grad nach rechts.
tt n	(turn to) setze Stift absolut in Richtung n Grad (Gradeinteilung siehe SETDRAW).
ma x,y	(move absolute) Bewege Stift (pu) auf absolute Position x/y (siehe auch SETDRAW).
da x,y	(draw absolute) Bewege Stift (pd) auf absolute Position x/y und zeichne dabei eine Linie.
mr x,y	(move relative) Bewege Stift (pu) auf Position x/y, relativ zur aktuelle Turtle-Position.
dr x,y	(draw relative) Bewege Stift (pd) auf Position x/y, relativ zur aktuellen Turtle-Position und zeichne dabei eine Linie.
co n	(color) Linienfarbe n einstellen (siehe COLOR).
pu	(pen up) Stift anheben (Stift schwebt).
pd	(pen down) Stift aufsetzen.

Eine DRAW-Zeile könnte folgendermaßen zusammengesetzt sein:

```
Draw "ma",X%,Y%,"tt",Int(10.52),"pd",A$,Len(B$),"lt60 fd3.4"
```

Die Angabe der Entfernungen, Winkel und Koordinaten etc. kann ebenfalls wahlweise als Ausdruck, als Konstante, als Variable oder innerhalb von Def$/"Def" erfolgen. Dabei ist die Anzahl der durch Komma getrennten Einzelanweisungen beliebig (max. Eingabezeilen-

länge = 255 Zeichen). Das Komma kann in den meisten Fällen auch vernachlässigt werden.

Beispiel:

```
Graphmode 3                          ! XOR-Modus
Yt%=Min(2,3-Xbios(4))                ! Y-Auflösungsteiler
For I%=0 To 640 Step 5               ! Einmal von links nach rechts
  For J%=0 To 1                      ! Zweimal hintereinander
    Setdraw 50+I%,200/Yt%+Cos(I%*Pi/180)*50,I%+90 ! Turtle setzen
    Draw "pd rt90 fd20 rt90 fd30 lt45 fd14.1 lt45 fd40"
    Draw "lt45 fd14.1 lt45 fd10 lt90 fd20 lt90 fd10 rt90"
    Draw "fd20 rt90 fd30 rt90 fd60 rt90 fd40 rt45 fd28.3"
    Draw "rt45 fd60 rt45 fd28.3 rt45 fd40"
    Draw "pu fd30 pd fd40 rt90 fd20 rt90 fd30 lt45 fd14.1"
    Draw "lt45 fd10 lt90 fd20 rt90 fd20 rt90 fd20 lt90 fd40"
    Draw "rt90 fd20 rt90 fd80 rt45 fd28.3 rt45 fd40"
    Draw "pu fd30 pd fd30 rt45 fd28.3 rt45 fd80 rt90 fd20"
    Draw "rt90 fd40 lt90 fd30 lt90 fd40 rt90 fd20 rt90 fd80"
    Draw "rt45 fd28.3 pu rt135 fd30 pd fd10 lt90 fd30 lt90"
    Draw "fd10 lt45 fd14.1 lt45 fd10 lt45 fd14.1 pu"
  Next J%
Next I%
```

Version 3.0

DRAW() — **Plotter-(Turtle-)Attribute liefern**

Var=DRAW(Index)

Liefert Informationen über die aktuellen DRAW-Turtle-Attribute.

Index:

```
0 = Aktuelle X-Position
1 = Aktuelle Y-Position
2 = Aktuell eingestellter Winkel (in Grad)
3 = Aktuelle X-Skalierung
4 = Aktuelle Y-Skalierung
5 = Pen-Flag (-1 = pd/0 = pu)
```

Mit Index 0 bis 4 werden Fließkommawerte geliefert.

Beispiel (Hires/Midres/Lowres):

Aus der Sprache LOGO kennen Sie vielleicht die Turtle (Schildkröte), die in den meisten Fällen als Pfeil dargestellt wird. Mit den folgenden beiden Prozeduren können Sie diese optische Turtle simulieren. Die Prozedur Showt speichert den aktuellen Hintergrund unter der Turtle und zeichnet einen Pfeil auf die aktuelle Position. Mit Hidet wird der Hintergrund wieder restauriert und die Turtle damit gelöscht.

```
Xt%=2-Sgn(Xbios(4))             ! X-Auflösung
Yt%=Min(2,3-Xbios(4))           ! Y-Auflösung
Setdraw 20,20,90                ! Turtle setzen
Print "TURTLE-Steuerung per Maustasten"
Showt                           ! Turtle zeigen
Do
  Dxmax=Max(15,Min(624/Xt%,Draw(0))) ! Immer schön..
  Dymax=Max(15,Min(384/Yt%,Draw(1))) ! ...im Bild...
  Setdraw Dxmax,Dymax,Draw(2) ! ...bleiben
  Hidet                         ! Turtle verstecken
  Draw "fd2"                    ! 2 Pixel zeichnen
  Showt                         ! Turtle zeigen
  If Mousek=1                   ! Linke Maustaste?
    Draw "lt2"                  ! Dann 2 Grad linksrum
  Else if Mousek=2              ! Rechte Maustaste?
    Draw "rt2"                  ! Dann 2 Grad rechtsrum
  Endif
Loop
'
Procedure Showt                 ! TURTLE-Proc
  Local Xt%,Yt%,Xtrtl,Ytrtl,Xtrtl2,Ytrtl2,Wtrtl,Ptrtl
  Xt%=2-Sgn(Xbios(4))           ! X-Auflösung
  Yt%=Min(2,3-Xbios(4))         ! Y-Auflösung
  Xtrtl=Draw(0)                 ! Aktuelle X-Position?
  Ytrtl=Draw(1)                 ! Aktuelle Y-Position?
  Xtrtl1=Max(0,Min(639/Xt%,Xtrtl-15)) !---. GET-
  Ytrtl1=Max(0,Min(399/Yt%,Ytrtl-15))     ! Bereich..
  Xtrtl2=Max(0,Min(639/Xt%,Xtrtl1+30))    ! eingrenzen
  Ytrtl2=Max(0,Min(399/Yt%,Ytrtl1+30))!---'
  Wtrtl=Draw(2)                 ! Aktueller Winkel?
  Ptrtl%=Draw(5)                ! Aktueller Pen-Status
  Get Xtrtl1,Ytrtl1,Xtrtl2,Ytrtl2,Trtlbckgrnd$
  '                             ! Hintergrund speichern
  Draw "pd lt120 fd11 rt150 fd20 rt120 fd20 rt150 fd11"
  '                             ! Turtle malen
  Setdraw Xtrtl,Ytrtl,Wtrtl     ! Alten Turtle-Status setzen
  If Ptrtl%=0                   ! Stift war vorher oben?
    Draw "pu"                   ! Dann wieder hochsetzen
  Endif
Return
'
Procedure Hidet                 ! Turtle löschen
  Put Xtrtl1,Ytrtl1,Trtlbckgrnd$
Return
```

LINE { LI } — Linie zeichnen

LINE X1,Y1,X2,X2

Die Bildschirm-Koordinaten X1,Y1 und X2,Y2 werden durch eine gerade Linie im aktuellen - zuletzt mit DEFLINE - eingestellten Linienstatus gezeichnet.

Beispiel:

```
For J%=1 To 20
  Line 10,J%*10,100,J%*10
  Line J%*10,10,J%*10,100
Next J%
```

PLOT { PL } — Punkt zeichnen

PLOT Xpos,Ypos

Xpos/Ypos bestimmt die Lage eines Bildschirm-Punktes, der gesetzt werden soll. PLOT wird von der Linienbreite-Definition durch DEFLINE nur beeinflußt, wenn Linien-Anfangs und -Endform als rund definiert wurden.

POINT() — Bildschirmpunkt-Farbwert ermitteln

Var=POINT(Xpos,Ypos)

Liefert die Nummer des Farbregisters, aus dem der durch Xpos/Ypos bezeichnete Bildschirm-Punkt seine Farbe bezieht (siehe COLOR).

Beispiel (Hires):

```
T%=Timer
Deffill ,2,2            ! DEFFILL hellgrau
Pbox 10,10,100,100      ! Kleine PBOX
For X%=10 To 100        ! Alle Punkte in X-Richtung
  For Y%=10 To 100      ! Alle Punkte in Y-Richtung
    If Point(X%,Y%)=0 ! Testen, ob nicht gesetzt
      ' If Ptst(X%,Y%)=0 ! Line-A-Test
      Plot 110+X%,Y%  ! Ja, dann setzen
    Endif
  Next Y%               ! Nächste Spalte
Next X%                 ! Nächste Zeile
Print (Timer-T%)/200'"Sek."
```

Anhand dieses Beispiels können Sie die POINT()- mit der PTST()-Funktion vergleichen. Lassen Sie es in der obigen Form laufen und aktivieren zum zweiten Durchlauf die If Ptst(..-Schleife.

Das Ergebnis des Programms ist das Negativ der zuerst gezeichneten PBOX, da statt jedes Punktes, der in der PBOX weiß ist, in der neuen Box ein Punkt gesetzt wird.

POLYLINE {POL} Polygon zeichnen

POLYLINE Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]

Die POLY-Befehle (POLYFILL, POLYLINE, POLYMARK) verwenden jeweils die Füll-, Linien- oder Marker-Attribute, die vorher mit DEFLINE, DEFFILL oder DEFMARK vorgenommen wurden.

In den Integer-Feldern Xp() und Yp() sind der Reihe nach so viele Koordinatenpaare anzugeben, wie Polygon-Ecken vorgesehen sind. Der erste zu definierende Punkt des Vielecks ist im Element 0 (OPTION BASE 0) bzw. im Element 1 (OPTION BASE 1) der Felder abzulegen, da die Inhalte der ersten beiden Feldelemente als Koordinatenwerte des ersten Punktes verwendet werden.

Die Anzahl der zu berücksichtigenden Polygon-Eckpunkte wird in Pkte übergeben. Maximal dürfen 128 Punkte angegeben werden. Soll mit POLYLINE ein geschlossener Linienzug gezeichnet werden, muß der letzte Punkt mit dem ersten identisch sein (Xp(n)=Xp(0)/Yp(n)=Yp(0)).

Um sich nicht jedesmal die Arbeit machen zu müssen, die Koordinatenpaare neu zu bestimmen, sobald dieselbe Figur an einer anderen Stelle gezeichnet werden soll, kann mit dem Zusatzbefehl OFFSET, der ggfs. den POLY-Befehlen angehängt wird, in X- und in Y-Richtung ein Versatz der Figur um einen bestimmten Betrag (negativ oder positiv) bewirkt werden.

Beispiel:

```
Dim X(10),Y(10)                   ! DIM Punkt-Felder
Yt%=Min(2,3-Xbios(4))             ! Y-Auflösung
Graphmode 3                       ! XOR-Modus
Do                                ! Endlos-Schleife
  For I=0 To 9                    ! 10 Punkte
    X(I)=Random(100)+56*I         ! Zufällige X-Koordinate
    Y(I)=Random(280/Yt%)+40/Yt%   ! Zufällige Y-Koordinate
  Next I
  X(I)=X(0)                       ! Letztes X = 1. X
  Y(I)=Y(0)                       ! Letztes Y = 1. Y
  Aa=Int(Rnd*5)+1                 ! Zufälliger Markertyp
  Bb=Int(Rnd*24)                  ! Zufälliges Füllmuster
  Cc=Random(5)+1                  ! Zufälliger Linientyp
  Defmark ,Aa,80                  ! Marker-DEF
  Deffill 1,2,Bb                  ! Muster-DEF
  Defline Cc,1,2,2                ! Linien-DEF
  Polymark 9,X(),Y()              ! Marker setzen
  Polyfill 9,X(),Y()              ! Fläche zeichnen
  For J%=1 To 2                   ! 2 mal
```

```
    For I%=3 To 99 Step 6          ! 16 Offsets
      Polyline 11,X(),Y() Offset I%,I% ! Vieleck zeichnen
    Next I%                        ! Nächstes Offset
  Next J%
  Polymark 9,X(),Y()               ! Marker löschen
  Polyfill 9,X(),Y()               ! Fläche löschen
Loop
```

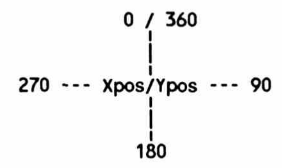
Version 3.0

SETDRAW { SETD } — DRAW-Turtle positionieren

SETDRAW Xpos,Ypos,Grad

Setzt den Stift für Plotter-(Turtle-)Grafik auf die absolute Position Xpos,Ypos in die Zeichenrichtung Grad (siehe auch DRAW $).

Grad:



Ein Beispiel hierzu finden Sie unter DRAW().

9.4 Line-A-Grafikbefehle

Version 3.0

ACHAR { AC } — Line-A-Einzel-Textzeichen ausgeben

ACHAR Ascii,Xpos,Ypos,Font,Art,Winkel

Gibt ein einzelnes Zeichen mit dem ASCII-Wert Ascii an der durch Xpos,Ypos bestimmten Position aus. Die Koordinatenangabe bezieht sich - im Gegensatz zu TEXT - dabei (in Normallage, d.h. Winkel = 0) auf die linke, obere Ecke der Zeichenbox, relativ zum absoluten Bildschirmnullpunkt.

Font bestimmt den zu verwendenden System-Font:

```
0 = 6x6     1 = 8x8     2 = 8x16
```

Wird ein größerer Wert für Font eingesetzt, wird dieser als Startadresse eines Font-Headers im GDOS-Format interpretiert (siehe VST_LOAD_FONTS()).

Art gibt die zu verwendende Textart an (siehe DEFTEXT).

Winkel enthält den gewünschten Textwinkel (siehe DEFTEXT).

Beispiel (Hires/Midres):

```
Aclip 1,0,0,639,199             ! Wichtig! Line-A-Clipping
For J%=0 To 31                  ! 32 Textarten
  For I%=0 To 2                 ! 3 Fonts
    For L%=0 To 2700 Step 900 ! 4 Winkel
      Cls                       ! Bildschirm löschen
      For K%=65 To 90           ! Zeichen A bis Z
        Achar K%,(K%-64)*23,20+J%*10+I%*16,I%,J%,L%
      Next K%
    Next L%
  Next I%
Next J%
```

Version 3.0

ACLIP { ACL } — Line-A-Clip-Box setzen

ACLIP Flag,X_li,Y_ob,X_re,Y_un

Begrenzt Line-A-Grafikausgaben (außer ALINE, HLINE, PSET und BITBLT) auf den hiermit beschriebenen Bildschirmausschnitt (vgl. CLIP). ACLIP ist - sofern die Gefahr besteht, daß Grafikausgaben über den Bildschirmrand hinausragen - bei Line-A Grafik grundsätzlich einzusetzen, da - anders als bei VDI-Ausgaben - die Grafik ohne Clipping bei Überschreitung des Bildschirmrandes direkt in die Nachbarbereiche des Bildschirmspeichers gezeichnet wird (evtl. fataler Fehler).

Flag gibt an, ob die gesetzte Clip-Box aktiv (Flag = 1) oder inaktiv (Flag = 0) sein soll.

X_li, Y_ob, X_re und Y_un sind die auf den absoluten Nullpunkt bezogenen Koordinaten der Clipping-Box.

Nach bisherigen Erfahrungen mit diesem Befehl bleibt er nach Aufruf nicht uneingeschränkt gültig. D.h., daß er evtl. durch Aufruf eines nachfolgenden Line-A-Befehls wieder aufgehoben wird. ACLIP sollte also immer unmittelbar vor einem Line-A-Grafikbefehl eingesetzt werden.

Version 3.0

ALINE { ALI } Line-A-Linie zeichnen

ALINE X1,Y1,X2,Y2,Farbe,Maske,Modus

ALINE ist prinzipiell identisch mit LINE, jedoch schneller. Die Koordinaten X1,Y1 und X2,Y2 werden mit einer geraden Linie in der angegebenen Farbe (siehe COLOR) verbunden.

Es muß zusätzlich in Maske ein 16-Bit-Wert übergeben werden, der das Linienmuster bestimmt (gesetztes Bit = Punkt) und in Modus der gewünschte Grafikmodus (0 - 3 entspricht GRAPHMODE 1 - 4).

GRAPHMODE hat auf Line-A-Befehle keine Wirkung. Alle Line-A-Koordinaten beziehen sich generell auf den absoluten Bilschirmnullpunkt.

Beispiel:

```
For I%=0 To 6000 Step 50          ! Umlauf-Schleife
  Xp1%=320+I%/20                  ! X-Mittelpunkt 1
  Xp2%=320-I%/20                  ! X-Mittelpunkt 2
  Xp3%=320+Sinq(I%)*310           ! X-Ellipsen-Punkt
  Yp%=100-Cosq(I%)*90             ! Y-Ellipsen-Punkt
  Aline Xp1%,100,Xp3%,Yp%,1,43690,1 ! Linie zeichnen
  Aline Xp2%,100,Xp3%,Yp%,1,43690,1 !   "      "
Next I%                           ! Nächster Grad-Schritt
```

Version 3.0

APOLY { AP } Line-A-Vieleck zeichnen

APOLY P_adr,P_anz,Ymin TO Ymax,Farbe,Modus,M_adr,M_anz

Es kann ein Vieleck bestimmt werden, dessen Figur (ohne Umrandung) anschließend in einem bestimmbaren Bereich mit einem vorgegebenen Muster gefüllt wird.

P_adr ist die Startadresse eines beliebig langen Word-Vektors, der abwechselnd die X- und die Y-Koordinaten (je zwei Byte = ein Word) der Vieleck-Punkte enthält. Durch P_anz wird angegeben, wieviele Punktkoordinaten (X/Y = 2 Words) aus diesem Vektor gelesen werden sollen.

Der Befehlsteil Ymin TO Ymax definiert eine oberste und eine unterste Bildschirmzeile, in deren Zwischenraum die Figur ausgefüllt wer-

den soll. Der Ausdruck TO kann auch durch ein Komma ersetzt werden.

Die Bedeutungen der Parameter Farbe, Modus, M_adr und M_anz sind mit denen der gleichnamigen Parameter bei HLINE identisch.

Dieser Befehl ist mit äußerster Vorsicht zu geniessen! Bei der überwiegenden Zahl von Testläufen war das Ergebnis ein Totalabsturz. Zu vermeiden sind diese Fehler nur durch peinlichst exakte Angaben der Koordinaten und des Füllbereichs.

APOLY ist bedingt mit POLYFILL vergleichbar und ca. um 10-15 Prozent schneller. Dieser Geschwindigkeitsvorteil schwindet allerdings dann, wenn keine häufigen Änderungen des Grafikmodus oder der Farbeinstellung notwendig sind. Wenn es darum geht, mehrere Polygone mit immer derselben Farbe (z.B. in Hires) und immer demselben Grafikmodus zu zeichnen, ist POLYFILL meiner Meinung nach gegenüber APOLY aufgrund des Komforts und der größeren Vielseitigkeit vorzuziehen.

Beispiel (Hires/Midres):

```
Dim P&(11)              ! 6 X- und 6 Y-Koordinaten
Deffill ,0              ! DEFFILL weiß
Padr%=V:P&(0)           ! Startadresse der Punktwords
M%=65535                ! Linienmuster = Voll-Linie
Madr%=V:M%              ! Startadresse des Linienmusters
Do                      ! Endlosschleife
  For I%=0 To 9         ! 10 Koordinaten..
    P&(I%)=Rand(200)    ! ..(abwechselnd X, Y) setzen
  Next I%               ! Nächste Koordinate
  P&(10)=P&(0)          ! Letzte = erste X-Koordinate
  P&(11)=P&(1)          ! Letzte = erste Y-Koordinate
  Pbox 19,19,241,171    ! Bereich löschen
  Aclip 1,20,0,240,199  ! Wichtig! Line-A-Clipping setzen
  Apoly Padr%,5,20 To 170,1,0,Madr%,1 ! APOLY-Aufruf
Loop
```

Version 3.0

ARECT { AR }   Line-A gefülltes Rechteck zeichnen

ARECT X1,Y1,X2,Y2,Farbe,Modus,M_adr,M_anz

Es wird ein - mit Muster gefülltes - Rechteck mit den absoluten Koordinaten X1/Y1 und X2,Y2 (= diagonal gegenüberliegende Ecken) gezeichnet.

Die Bedeutungen der Parameter Farbe, Modus, M_adr und M_anz sind mit denen der gleichnamigen Parameter bei HLINE identisch.

ARECT ist prinzipiell identisch mit PBOX ohne Rahmen. Auch hier ist - wie bei APOLY - der Geschwindigkeitsvorteil nur dann ausschlaggebend, wenn Farb- oder Grafikmodus-Änderungen notwendig werden. Wenn man dann durch BOUNDARY 0 den PBOX-Rahmen ausschaltet, liegt der Unterschied bei nur noch ca. einem Prozent.

Es kann übrigens fatale Folgen haben, wenn die Koordinatenwerte der linken, oberen Ecke (X1/Y1) größer als die der rechten, unteren Ecke (X1/Y2) werden.

Beispiel:

```
Aclip 1,0,0,319,199        ! Wichtig! Line-A-Clipping setzen
For I%=0 To 100
  M&=I%                    ! Füllmuster = Laufindex
  Madr%=V:M&               ! Startadresse des Füllmusters
  Arect 10,20+I%,500,80+I%,1,2,Madr%,1
Next I%
```

Version 3.0

ATEXT { AT } — Line-A Grafiktext ausgeben

ATEXT Xpos,Ypos,Font,Text$

Gibt den String Text$ an der durch Xpos,Ypos bestimmten Position aus. Die Koordinatenangabe bezieht sich dabei auf die linke obere Ecke der Zeichenbox des ersten Zeichens, relativ zum absoluten Bildschirmnullpunkt.

Font bestimmt den zu verwendenden System-Font:

```
0 = 6x6   1 = 8x8   2 = 8x16
```

Auch hier wird - wie bei ACHAR - ein größerer Wert für Font als Startadresse eines Font-Headers im GDOS-Format interpretiert (siehe VST_LOAD_FONTS()).

Bei ATEXT kann im Gegensatz zu TEXT **kein** beliebiger Schriftstil oder Textwinkel angegeben werden. Beschränkt man den Vergleich zwischen beiden Textausgaben jedoch nur auf die Schriftgrößen DEFTEXT ,,,4, DEFTEXT ,,,6 und DEFTEXT ,,,13 - welche den hier anzugebenden Fonts 0, 1 und 2 entsprechen - so ist ATEXT um ca. 6 - 10 Prozent schneller.

Beispiel (Hires/Midres/Lowres):

```
Aclip 1,0,0,319,199         ! Wichtig! Line-A-Clipping setzen
For I%=0 To 2               ! 3 Fonts
  Atext 10+I%*20,60+I%*12+I%*20,I%,"Font "+Str$(I%)
Next I%
```

Version 3.0

HLINE {HL} **Line-A horizontale Linie zeichnen**

HLINE X1,Y,X2,Farbe,Modus,M_adr,M_anz

HLINE ist prinzipiell identisch mit LINE, jedoch **nur** für waagerechte Linien. Außerdem ist HLINE erheblich schneller (ca. 35 - 45 Prozent). Die X-Koordinaten X1 und X2 in der Zeile Y werden mit einer Linie in der angegebenen Farbe (Hardware-Register siehe SETCOLOR) verbunden.

In Modus wird der gewünschte Grafikmodus (0-3 entspricht GRAPHMODE 1 - 4) übergeben. Zusätzlich muß in M_adr die Startadresse eines Speicherblocks angegeben werden. Durch M_anz kann bestimmt werden, wieviele 16-Bit-Werte ab dieser Adresse als Linienmuster gewertet werden sollen (Anzahl der Muster minus 1 = M_anz).

Für die erste gezeichnete Linie wird das erste Word als Muster genommen (gesetztes Bit = Punkt), für die nächste Linie das nächste Word usw. Wurden 'Anz' Muster gelesen, beginnt die Zählung wieder beim ersten Word usw. Auch wenn bei mehreren HLINE-Aufrufen dieselbe Adresse angegeben wird, wird der in 'M_anz' verfügte Offset-Rythmus beibehalten.

Beispiel: Dieses Beispiel zeigt, daß durch HLINE auch PBOX-Simulationen mit frei definierbarem Füllmuster möglich sind. Das gleiche gilt übrigens auch für ARECT. Dabei ist das Raster des Füllmusters zwar in der Breite auf 16 Punkte beschränkt, jedoch nicht in der Höhe.

```
Deffill ,2,1                !-----. Füllmuster
Pbox 0,0,15,34                    !
Deftext ,1,900,6                  !- Füllmuster vorbereiten
Text 10,32,"FILL"                 !
Get 0,0,15,34,M$            !-----'
Madr%=V:M$+6                ! Startadresse des Füllmusters
Aclip 1,0,0,319,199         ! Wichtig! Line-A-Clipping setzen
For I%=20 To 180
  Hline 20,I%,300,1,0,Madr%,31
Next I%
Box 20,20,300,180
```

Version 3.0

PSET { PS } Line-A-Punkt zeichnen

PSET Xpos,Ypos,Farbe

PSET ist identisch mit PLOT (siehe dort), jedoch erheblich schneller (mehr als dreimal so schnell!). Anders als bei PLOT muß hier zusätzlich zu den Koordinaten in Farbe die Punktfarbe angegeben werden (siehe COLOR). GRAPHMODE hat auf Line-A-Befehle keine Wirkung.

Beispiel (Hires/Midres/Lowres): Weil mir als Beispiel für diesen Befehl kein "berauschender" Effekt einfallen wollte, finden Sie hier - um das gezeichnete Punktraster sinnvoll werden zu lassen - eine Raster-Abfrageroutine.

Diese Erweiterung habe ich speziell für grafische Menüs entwickelt. So können Grafik-Icons, die in einem derartigen Raster angeordnet sind, leicht angewählt werden. Durch die Begrenzungsmöglichkeiten mit Xmin%, Ymin%, Xmax% und Ymax% kann der Auswahlbereich je nach Bedarf eingeschränkt werden.

```
For J%=10 To 300 Step 8          !-------.
  For I%=10 To 170 Step 8                !
    Pset J%,I%,1                         !- Raster zeichnen
  Next I%                                !
Next J%                          !-------'
Setmouse 150,100,0               ! Maus in das Raster setzen
'
'       Die folgende kleine Routine simuliert für
'       die V2.xx-Versionen den SETMOUSE-Befehl.
'       '
'       @Setmouse(150,100,0)      ! Aufruf
'       '
'       Procedure Setmouse(Xpos%,Ypos%,Mbut%)
'         Xt%=2-Sgn(Xbios(4))    ! X-Auflösungsteiler
'         Yt%=Min(2,3-Xbios(4)) ! Y-Auflösungsteiler
'         Xpos%=Max(0,Min(639/Xt%,Xpos%)) ! X begrenzen
'         Ypos%=Max(0,Min(399/Yt%,Ypos%)) ! Y begrenzen
'         Mbut%=Max(0,Min(3,Mbut%))  ! Status begrenzen
'         Dpoke 9952,Xpos%        ! Maus-X-Position setzen
'         Dpoke 9954,Ypos%        ! Maus-Y-Position setzen
'         Dpoke 9958,Mbut%        ! Maustasten-Status setzen
'       Return
'
Text 30,185,"Gesamtes Feld ist wählbar"
@Raster(3,10,10,8,8,1,1,36,20,36,20,*Index%,*Dummy%)
If Index%=0
  Text 30,195,"Raster ohne Klick verlassen     "
Else
  Text 30,195,"gewählt: Feld "+Str$(Index%)+"/"+Str$(Dummy%)
```

```
Endif
Text 10,185,"Nur Reihe 3 ist wählbar  "
@Raster(0,10,10,8,8,3,1,3,20,36,20,*Index%,*Dummy%)
Text 10,195,"gewählt: Feld "+Str$(Index%)+"  "
Pause 20
Text 10,185,"Nur Zeile 14 ist wählbar  "
@Raster(0,10,10,8,8,1,14,36,14,36,20,*Index%,*Dummy%)
Text 10,195,"gewählt: Feld "+Str$(Index%)+"  "
Pause 20
Text 10,185,"Nur Teilfeld 3/3 - 16/16 ist wählbar"
@Raster(0,10,10,8,8,3,3,16,16,36,20,*Index%,*Dummy%)
Text 10,195,"gewählt: Feld "+Str$(Index%)+"  "
Pause 20
Text 10,185,"Nur Einzelpunkt 36/20 ist wählbar    "
@Raster(0,10,10,8,8,36,20,36,20,36,20,*Index%,*Dummy%)
Text 10,195,"gewählt: Ende         "
'
Procedure Raster(Rflg%,Rxl%,Ryl%,Rxb%,Ryb%,Xmin%,Ymin%,...
                 ...Xmax%,Ymax%,Xfld%,Yfld%,R.inx%,R.iny%)
  '
  ' Mausraster-Index-Ermittlung (alle Auflösungen)
  '
  ' Rflg%:  Flag
  '         0 = Die Routine wird nur durch Mausklick auf
  '             eines der Rasterfelder verlassen. Es wird
  '             ein Kettenindex nur in R.inx% zurückgegeben
  '             (siehe unten bei R.inx%).
  '         1 = Wie 0, jedoch wird die Routine auch verlassen,
  '             wenn der Mauszeiger außerhalb des angegebenen
  '             Rasterbereichs (Xmin%, Ymin%, Xmax%, Ymax%)
  '             gebracht wird. In diesem Fall wird in R.inx%
  '             eine Null zurückgegeben. Bei Aufruf dieses Modus
  '             sollte allerdings sichergestellt werden, daß
  '             sich der Mauszeiger innerhalb des Rasters befindet.
  '             Bei Mausklick wird ein Kettenindex nur in
  '             R.inx% zurückgegeben (siehe unten bei R.inx%).
  '         2 = Wie 0, jedoch wird in R.inx% kein Kettenindex
  '             geliefert, sondern in R.inx% der X-Index und in
  '             R.iny% der Y-Index des gewählten Rasterpunktes
  '             geliefert.
  '         3 = Wie 1, jedoch wird in R.inx% kein Kettenindex
  '             geliefert, sondern in R.inx% der X-Index und in
  '             R.iny% der Y-Index des gewählten Rasterpunktes
  '             geliefert.
  ' Rxl%:   X-Koordinate des gesamten Rasterfeldes
  ' Ryl%:   Y-Koordinate des gesamten Rasterfeldes
  ' Rxb%:   Breite eines Rasterpunktes in Pixel
  ' Ryb%:   Höhe eines Rasterpunktes in Pixel
  ' Xmin%:  Kleinste wählbare Rasterreihe (min. 1 = 1. Reihe)
  ' Ymin%:  Kleinste wählbare Rasterzeile (min. 1 = 1. Zeile)
  ' Xmax%:  Größte wählbare Rasterreihe (max. Xfld%)
  ' Ymax%:  Größte wählbare Rasterzeile (max. Yfld%)
  ' Xfld%:  Anzahl der waagerechten Rasterpunkte des Gesamtfeldes
  ' Yfld%:  Anzahl der senkrechten Rasterpunkte des Gesamtfeldes
  ' R.inx%:
  '        - Bei Rflg% 0 oder 1 = gewählter Rasterindex (verkettet)
  '
  '         Wird z.B. ein Gesamtraster von z.B. 5*3 Punkten angegeben,
  '         so ergibt sich der Index der Punkte nach folgendem
```

```
'         Schema (Kettenindex):
'
'
'              .----.----.----.----.----.
'              | 1  | 2  | 3  | 4  | 5  |
'              |----|----|----|----|----|
'              | 6  | 7  | 8  | 9  | 10 |
'              |----|----|----|----|----|
'              | 11 | 12 | 13 | 14 | 15 |
'              !----!----!----!----!----!
'
'         Wird der gewählte Punkt mit der linken Maustaste
'         angeklickt, erhält man den positiven Index. Bei Klick
'         mit der rechten Maustaste erhält man eine Null
'         (= Abbruch) und bei Klick mit beiden Maustasten
'         gleichzeitig erhält man den negativen Index.
'
'       - Bei Rflg% 2 oder 3 = X-Index des Punktes.
'
' R.iny%: Nur bei Rflg% 2 oder 3 = Y-Index des Punktes
'
'         Im obigen Beispielraster ergäbe sich bei Rflg% 2
'         oder 3 folgende Indizierung (X/Y):
'
'
'              .----.----.----.----.----.
'              |1/1 |2/1 |3/1 |4/1 |5/1 |
'              |----|----|----|----|----|
'              |1/2 |2/2 |3/2 |4/2 |5/2 |
'              |----|----|----|----|----|
'              |1/3 |2/3 |3/3 |4/3 |5/3 |
'              !----!----!----!----!----!
'
'
' Die Routine verändert COLOR (1) und GRAPHMODE (1).
' Nach Rückkehr müssen diese Einstellungen ggfs. restauriert
' werden.
'
Local Msx%,Msy%,K%,Ayp%,Axp%,Axp2%,Ayp2%,Msxx%,Msyy%,R.back$
Xmin%=Max(1,Xmin%)          ! Xmin% = mind. 1
Ymin%=Max(1,Ymin%)          ! Ymin% = mind. 1
Xmax%=Min(Xfld%,Xmax%)      ! Xmax% = max. Xfld%
Ymax%=Min(Yfld%,Ymax%)      ! Ymax% = max. Yfld%
Rflg%=Max(0,Min(3,Rflg%))   ! Rflg% auf den Bereich
'                           ! 0 bis 3 eingrenzen
Color 1                     ! Zeigerbox-Farbe (kann im
' Farbbetrieb auch anders sein, jedoch <> Hintergrundfarbe)
Graphmode 1                 ! XOR-Modus
Hidem                       ! Maus ausschalten
Repeat                      ! Zeiger-Schleife
  Mouse Msx%,Msy%,K%        ! Maus-Status holen
  '
  Axp%=Max(Xmin%-1,Min(Xmax%-1,Int((Msx%-Rxl%)/Rxb%)))
  ' Mausposition in Raster-Index umrechnen
  Ayp%=Max(Ymin%-1,Min(Ymax%-1,Int((Msy%-Ryl%)/Ryb%)))
  '
  Msx%=Rxl%+Axp%*Rxb%
  ' Raster-Index in skalierte Mausposition umrechnen
  Msy%=Ryl%+Ayp%*Ryb%
  '
  Get Msx%-1,Msy%-1,Msx%+Rxb%+1,Msy%+Ryb%+1,R.back$
  ' Zeigerbox-Hintergrund sichern
```

```
    Box Msx%-1,Msy%-1,Msx%+Rxb%+1,Msy%+Ryb%+1 ! Zeiger...
    Box Msx%,Msy%,Msx%+Rxb%,Msy%+Ryb%         ! ...zeichnen
    Repeat                       ! Auf Mausbewegung warten  >-----.
      Mouse Msxx%,Msyy%,K%       ! Maus-Status holen
      If Rflg%=0 Or Rflg%=2      ! Mausklick-Exit?
        '
        Axp2%=Max(Xmin%-1,Min(Xmax%-1,Int((Msxx%-Rxl%)/Rxb%)))
        ' neuen Index berechnen (nur innerhalb des Rasters)
        Ayp2%=Max(Ymin%-1,Min(Ymax%-1,Int((Msyy%-Ryl%)/Ryb%)))
      Else                       ! auch "außerhalb"-Exit!
        Axp2%=Int((Msxx%-Rxl%)/Rxb%)
        ' neuen Index berechnen (auch außerhalb des Rasters)
        Ayp2%=Int((Msyy%-Ryl%)/Ryb%)
        '
      Endif
      @Doing                     ! Irgend etwas tun?! (s.u.)
    Until Axp%<>Axp2% Or Ayp%<>Ayp2% Or K% ! Maustaste   <------'
    '                            ! gedrückt oder Index verändert?
    Put Msx%-1,Msy%-1,R.back$    ! Hintergrund restaurieren
  Until K% Or ((Rflg%=1 Or Rflg%=3) And (Axp2%<0...
       ...Or Axp2%>Xmax%-1 Or Ayp2%<0 Or Ayp2%>Ymax%-1))
  ' Diese UNTIL-Zeile ist aus drucktechnischen Gründen getrennt
  ' (siehe auch Prozedurkopf). Die drei Trennpunkte gehören nicht
  ' zur Zeilensyntax, sondern sollen nur die Zeilentrennung
  ' verdeutlichen.
  Showm                          ! Maus wieder zeigen
  If Rflg%=0 Or Rflg%=1          ! Kettenindex gefragt?
    If K%                        ! Maustaste gedrückt?
      *R.inx%=(Ayp2%*(Xfld%)+Axp2%+1)*Sgn(2-K%)
      '                          ! Kettenindex zurückgeben
    Else                         ! Zeiger außerhalb des Rasters!
      *R.inx%=0                  ! Index%=0
    Endif
  Else                           ! X/Y-Index gefragt!
    If K%                        ! Maustaste gedrückt?
      *R.inx%=Axp2%+1            ! X-Index zurückgeben
      *R.iny%=Ayp2%+1            ! Y-Index zurückgeben
    Else                         ! Zeiger außerhalb des Rasters!
      *R.inx%=0                  ! X-Index = 0
      *R.iny%=0                  ! Y-Index = 0
    Endif
  Endif
Return
'
Procedure Doing
  ' Diese Prozedur muß vorhanden sein, wenn Raster aufgerufen
  ' wird und der dortige Doing-Aufruf ausgeführt werden soll.
  ' Wie hier gezeigt, kann in dieser Prozedur beliebiger Text
  ' ausgegeben, auf bestimmte Rasterpunkte reagiert und/oder z.B.
  ' die Tastatur bzw. Ereignisse überwacht werden.
  '
  ' On menu                      ! Evtl. Ereignis-Überwachung
  Pt%=(Ayp2%*(Xfld%)+Axp2%+1)*Sgn(2-K%) ! Kettenindex..
  Text 230,195,"Punkt: "+Str$(Pt%)+"  " ! ...ausgeben
Return
```

Version 3.0

PTST — Line-A-Bildschirmpunkt testen

Var=PTST(Xpos,Ypos)

PTST ist identisch mit POINT(), jedoch unter günstigen Umständen ca. doppelt so schnell. Im Durchschnitt ist POINT() um umgefähr 50 Prozent langsamer. Beachten Sie dazu bitte den POINT()/PTST()-Vergleich bei POINT().

9.5 Grafikoperationen

Version 3.0

CLIP { CLI } — Grafikausgabe begrenzen/Nullpunkt setzen

CLIP Xpos,Ypos,Breite,Höhe [OFFSET X,Y]
CLIP Xl,Yo TO Xr,Yu [OFFSET X,Y]
CLIP #Windownummer [OFFSET X,Y]
CLIP OFFSET X,Y
CLIP OFF

Ermöglicht die Bestimmung eines Bildschirmrechtecks, auf welches dann sämtliche Grafikausgaben begrenzt werden (außer PUT, LINE-A-Grafik, BITBLT und AES-Grafik) bzw. die Festlegung des Koordinaten-Nullpunktes für Grafikausgaben. Alle Teile von VDI-Grafikausgaben (LINE, BOX, PBOX, DRAW etc.), die die Grenzen dieses Rechtecks überschreiten, werden an dessen Grenzen abgeschnitten.

Syntax-Variante 1: Xpos und Ypos beschreiben die Position, sowie Breite und Höhe die Maße des CLIP-Rechtecks.

Syntax-Variante 2: Xl und Yo beschreiben die Position der linken, oberen Ecke des CLIP-Rechtecks. Xr und Yu (hinter TO) beschreiben die Position seiner rechten, unteren Ecke.

Syntax-Variante 3: #Windownummer enthält die GFA-Fensternummer, dessen Arbeitsbereichs-Position und -Ausmaße für das CLIP-Rechteck gelten sollen.

Syntax-Variante 4: X und Y legen den Koordinaten-Ursprung (in Relation zur linken, oberen Bildschirmecke) für künftige Grafikausgaben fest (außer PUT, LINE-A-Grafik, BITBLT und AES-Grafik). OFF-

SET X,Y kann auch optional an die vorangegangenen CLIP-Befehlsvarianten angehängt werden, wodurch ebenfalls der Grafik-Nullpunkt bestimmt wird.

Syntax-Variante 5: CLIPP OFF schaltet aktuelles Clipping wieder aus.

Beispiel zu 1, 2, 4 und 5:

```
Deffill ,2,4             ! DEFFILL grau
Pbox 2,2,218,118         ! Hintergrund-Fläche zeichnen
Deffill ,2,2             ! DEFFILL hellgrau
Clip 10,10,100,100       ! Ab 10/10 100 Pixel in jeder
'                        ! Richtung als Clip-Box
Pcircle 60,60,60         ! Gefüllten Kreis zeichnen
Clip 110,10 To 210,110 Offset 100,10
' Bildschirmbereich mit den Koordinaten 110/10 und 210/110
' als Clip-Bereich anmelden und Nullpunkt auf 100/10 setzen.
Pcircle 60,50,60         ! Gefüllten Kreis zeichnen
Clip Off                 ! Clipping ausschalten
Clip  Offset 0,0         ! Grafik-Nullpunkt wieder auf 0/0 setzen
Box 10,10,210,110        ! Box um die beiden Kreis-Ausschnitte  '
Procedure Clip(Clxl%,Clyo%,Clxr%,Clyu%)
  '
  ' 'CLIP Xl,Yo TO Xr,Yu'-Simulation für V2.xx
  ' ---------------------------------------------
  ' Auch hier wieder eine V2.xx-Erweiterung mit dem Ziel,
  ' V3.0-Befehle auch in V2.xx einsetzbar zu machen, wodurch
  ' die spätere Anpassung der V2xx-Listings an die V3.0-Syntax
  ' - spätestens bei Erscheinen des V3.0-Compilers - wesentlich
  ' erleichtert wird.
  '
  ' Clxl% = Linke X-Koordinate des Ausschnitts
  ' Clyo% = Obere Y-Koordinate des Ausschnitts
  ' Clxr% = Rechte X-Koordinate des Ausschnitts
  ' Clyu% = Untere Y-Koordinate des Ausschnitts
  '
  Local Cl.buff$,Handle%
  Handle%=Dpeek(Vdibase+40)          ! GFA-Handle holen
  Cl.buff$=Mkl$(2)+Mkl$(1)+Mkl$(0)+Mki$(Handle%)  !---. CONTROL-Feld
  Bmove Varptr(Cl.buff$),Contrl,14                !---' (VDI) belegen
  Cl.buff$=Mki$(Clxl%)+Mki$(Clyo%)+Mki$(Clxr%)+Mki$(Clyu%)
  Bmove Varptr(Cl.buff$),Ptsin,8     ! PTSIN-Feld belegen
  Dpoke Intin,1                      ! Clip-Flag (1 = an)
  Vdisys 129
Return
'
Procedure Clip_off
  '
  ' 'CLIP OFF-Simulation für V2.xx
  '
  Local Xt%,Yt%,Cxr%,Cyu%
  Xt%=2-Sgn(Xbios(4))                ! X-Auflösungsteiler
  Yt%=Min(2,3-Xbios(4))              ! Y-Auflösungsteiler
  Cxr%=639/Xt%                       ! Maximale Bildschirmbreite
  Cyu%=399/Yt%                       ! Maximale Bildschirmhöhe
  Clip(0,0,Cxr%,Cyu%)                ! Prozedur Clip aufrufen
```

```
Return
'
Procedure Clip(Windownummer%)
  '
  ' CLIP #Windownummer-Simulation für V2.xx
  '
  Local Cxl%,Cyo%,Cxr%,Cyu%
  Cxl%=Dpeek(Windtab+4+12*(Windownummer%-1)) !--------.
  Cyo%=Dpeek(Windtab+6+12*(Windownummer%-1))          ! Window-
  Cxr%=Cxl%+Dpeek(Windtab+8+12*(Windownummer%-1))     ! Maße
  Cyu%=Cyo%+Dpeek(Windtab+10+12*(Windownummer%-1)) !--' holen
  Clip(Cxl%,Cyo%,Cxr%,Cyu%)      ! Prozedur Clip aufrufen (s.o.)
Return
```

Die CLIP OFFSET X0,Y0-Variante läßt sich für die V2.xx-Versionen leider nicht innerhalb einer Prozedur realisieren. Vor Öffnen eines Ausgabe-Fensters kann man dies durch OPENW 0,X0,Y0 erreichen (siehe unter OPENW). Allerdings ist dies nicht möglich, solange irgendwelche Fenster geöffnet sind, da sonst die Fensterbereiche durcheinander geraten. Bei Aufruf von OPENW 0 darf also kein Fenster geöffnet sein.

GET — Bildschirmbereich speichern

GET X_links,Y_oben,X_rechts,Y_unten,Var$

Durch GET (nicht zu verwechseln mit dem Diskettenbefehl GET#) wird mit den Koordinatenangaben X_links/Y_oben und X_rechts/Y_unten ein Bildschirmausschnitt definiert, welcher als Bitmuster in Var$ eingelesen wird. Die angegebenen Eck-Koordinaten müssen innerhalb des Bildschirmbereichs liegen, da sonst der Interpreter den Befehl nicht ausführt.

Diese sparsame Beschreibung läßt nicht ahnen, was mit diesem Befehl alles machbar ist. In Version V3.0 bietet dieser Befehl in Verbindung mit RC_COPY geradezu eine optimale Grundlage für die gepflegte und schnelle Animation von Bit-Images. Aber auch in V2.xx ist dies einer der wichtigsten Befehle überhaupt.

Beispiel: Und wieder mal ein "kleines" Demo-Listing. Es produziert eines der so heiß begehrten Pop-Up-Menüs. Dies sind Menüs, die nicht - wie vom GEM her gewohnt - vom oberen Bildrand "herunterfallen" bzw. "heruntergezogen" werden (Drop-Down-, bzw. Pull-Down-Menü), sondern an der Bildschirmposition erscheinen, an welcher sie benötigt werden, nämlich unter dem Mauszeiger.

Dabei kann ein beliebiger Bildschirmbereich angegeben werden, innerhalb dessen das Menü dargestellt werden soll, bzw. der als Reaktionsbereich fungiert. Ist der Mauszeiger zum Zeitpunkt des Aufrufs innerhalb dieses Bereichs, so bleibt das Menü unter dem Mauszeiger stehen. Wird ein Menüpunkt angewählt oder verläßt der Mauszeiger den Bereich wieder (auch ohne Auswahl), so wird das Menü wieder gelöscht. Unter welchen Bedingungen Sie den Menü-Aufruf zulassen, bleibt Ihnen überlassen.

Bei der Einrichtung des Menüs habe ich versucht, weitestgehend den Konventionen der GFA-Pull-Down-Menü-Definition Rechnung zu tragen. So wird z.B. auch hier ein Menüzeilentext, der mit einem Bindestrich beginnt, als unwählbar (grau) dargestellt. Der Punkt ist dann von der Auswahl ausgeschlossen.

Ein wesentlicher Unterschied ist, daß hier aus einem DATA-Block, der die einzelnen Zeilentexte enthält, über einen wählbaren Startindex beliebige Menüteile ausgeschnitten und angezeigt werden können. Außerdem ist die Anzeige des Menüzeilentextes auf eine Pixel-Breite von 100 Pixel (Hires/Midres) bzw. 50 Pixel (Lowres) beschränkt. Der Text wird per TEXT auf diese Breite formatiert und in den Menüpunkt eingesetzt.




```
Xt%=2-Sgn(Xbios(4))           ! X-Auflösungsteiler
Yt%=Min(2,3-Xbios(4))         ! Y-Auflösungsteiler
Start%=1                      ! Erst ab 2. Menüpunkt darstellen
Anzahl%=21                    ! Insgesamt 6 Menüpunkte darstellen
Dim Feld$(Start%+Anzahl%)     ! Textfeld einrichten. Nur so viele
'                   Menüpunkt-Elemente, wie zur Demo nötig sind.
Restore M.datas               ! Datazeiger setzen
For I%=1 To Start%+Anzahl%    ! Menütext-Datas
  Read Feld$(I%)              ! Einlesen
Next I%
```

```
M.datas:
Data ---,Speichern,Laden,Löschen,Kopieren,----------,QUIT
Data ---,Speichern,Laden,Löschen,Kopieren,----------,QUIT
Data ---,Speichern,Laden,Löschen,Kopieren,----------,QUIT
```

Der Einfachheit halber habe ich hier dreimal dieselben Texte verwendet, um den folgenden Block der IF..ENDIF-Abfragen in Grenzen zu halten. In V3.0 können diese Abfragen auch leicht durch IF..ELSE IF oder SELECT..CASE vereinfacht werden.

Im folgenden Listing mußten wieder einige Programmzeilen aus drucktechnischen Gründen getrennt werden. Die drei Trennpunkte am Zeilenende und Zeilenbeginn gehören nicht zur Zeilensyntax.

```
Box 50/Xt%,50/Yt%,300/Xt%,300/Yt%
Do
  Repeat
    If Mousex>50/Xt% And Mousex<300/Xt% And Mousey> ...
                          ... 50/Yt% And Mousey<399/Yt%
      @Menue(Start%,Anzahl%,50/Xt%,50/Yt%,300/Xt%, ...
                          ... 399/Yt%,*Feld$(),*Index%)
    Endif
    ' Weitere Aufruf-Variante:
    ' If Mousek=2
    '   @Menue(Start%,Anzahl%,0,0,639/Xt%,399/Yt%, ...
    '                              ... *Feld$(),*Index%)
    ' Endif
  Until Index%>0                ! Or Mousek
  If Feld$(Index%+Start%)="Speichern"
    Feld$(Index%+Start%)="-Speichern"
    Feld$(Index%+Start%+1)="Laden"
    Print At(1,1);"gewählt: Speichern      "
  Endif
  If Feld$(Index%+Start%)="Laden"
    Feld$(Index%+Start%-1)="Speichern"
    Feld$(Index%+Start%)="-Laden"
    Print At(1,1);"gewählt: Laden          "
  Endif
  If Feld$(Index%+Start%)="Löschen"
    Print At(1,1);"gewählt: Löschen        "
  Endif
  If Feld$(Index%+Start%)="Kopieren"
    Feld$(Index%+Start%)=Chr$(8)+" Kopieren"
    Print At(1,1);"gewählt: Kopieren       "
    Goto Label
  Endif
  If Feld$(Index%+Start%)=Chr$(8)+" Kopieren"
    Feld$(Index%+Start%)="Kopieren"
    Print At(1,1);"gewählt: Kopieren       "
  Endif
  If Feld$(Index%+Start%)="QUIT"
    Print At(1,1);"gewählt: Quit           "
  Endif
  Label:
  Print "Menüindex : ";Index%
  Exit if Feld$(Index%+Start%)="QUIT"
```

```
  Clr Index%
Loop
'
Procedure Menue(Pm1%,Mmx%,Mxl%,Myo%,Mxr%,Myu%,F.adr%,V.adr%)
  ' Pop-Up-Menü (Hires/Midres/Lowres)
  '
  ' Pm1%   = Index des Menüpunktes (Textfeldindex), der an
  '          erster Stelle erscheinen soll. Die Texte müssen
  '          ab Index 0 im Textfeld stehen.
  ' Mmx%   = Anzahl der Menüpunkte, die ab Pm1% dargestellt
  '          werden sollen. Der Menü-Index liegt immer im
  '          Bereich von 1 bis Mmx%.
  ' Mxl%, Myo%, Mxr%, Myu%
  '        = Rahmenkoordinaten, innerhalb derer das Menü
  '          dargestellt werden soll. Paßt das Menü nicht
  '          in den Rahmen, liegt die rechte, untere Ecke
  '          des Menüs immer auf der rechten, unteren Ecke
  '          des Rahmens.
  ' F.adr% = Pointer auf das Textfeld.
  ' V.adr% = Pointer auf eine Rückgabevariable.
  '          Die Rückgabevariable enthält nach Abschluß den
  '          Index des gewählten Menüpunktes. Um den dazugehörigen
  '          Textfeld-Index zu ermitteln, muß zum Menüindex 'Pm1%'
  '          addiert werden.
  Local Mmen$,Msk%,M.key$,Yi%,Yi2%,Mrs%,M.i%,Lsr%
  Local Mxl2%,Mxr2%,Myo2%,Myu2%,M.xt%,M.yt%
  M.xt%=2-Sgn(Xbios(4))          ! X-Auflösungsteiler
  M.yt%=Min(2,3-Xbios(4))        ! Y-Auflösungsteiler
  Dim Dum$(1)                    ! Lokales Swap-Feld
  Swap *F.adr%,Dum$()            ! Felder swappen
  Mxl2%=Min(Max(Mousex-(68/M.xt%),Mxl%),Mxr%-(136/M.xt%))
  Mxr2%=Mxl2%+(136/M.xt%)
  Myo2%=Min(Max(Mousey-(6/M.yt%),Myo%),Myu%-(18+Mmx%*18)/M.yt%)
  Myu2%=Myo2%+((18+Mmx%*18)/M.yt%)
  Mxl%=Min(Mxl%,Mxl2%)
  Myo%=Min(Myo%,Myo2%)
  '
  ' Die letzten 6 Zeilen haben die schwierige Aufgabe, den
  ' gewünschten Darstellungsbereich des Menüs und die Position
  ' der Maus so miteinander zu verknüpfen, daß das Menü -
  ' wenn möglich - unter dem Mauszeiger, aber nicht außerhalb
  ' des gewählten Rahmens dargestellt wird.
  '
  Get Max(0,Mxl2%),Max(0,Myo2%),Min((639/M.xt%), ...
                ... Mxr2%),Min(Myu2%,399/M.yt%),Mmen$
  ' Menü-Hintergrund speichern
  Deftext 1,0,0,8-2*M.yt%        ! Textgröße anpassen
  Deffill 1,0,0                  ! DEFFILL weiß
  Graphmode 1                    ! Replace-Modus
  Pbox Mxl2%,Myo2%,Mxr2%,Myu2% !----------------------.
  Box Mxl2%+(1/M.xt%),Myo2%+(1/M.yt%),Mxr2%-(1/ ...   !
                         ... M.xt%),Myu2%-(1/M.yt%)    !
  Deffill 1,2,4                  ! DEFFILL grau        !
  Pbox Mxl2%+(6/M.xt%),Myo2%+(6/M.yt%),Mxr2%-(6/ ...   !
                         ... M.xt%),Myu2%-(6/M.yt%)    !
  Deffill 1,0,0                  ! DEFFILL weiß        !
  For M.i%=1 To Mmx%             ! Alle Menü-Zeilen    !
    Graphmode 1                  ! Replace-Modus       !
    Pbox Mxl2%+(13/M.xt%),Myo2%-6/M.yt%+M.i%*18/ ...   !
```

```
    ... M.yt%,Mxr2%-(13/M.xt%),Myo2%+6/M.yt%+M.i%*18/M.yt%
   Graphmode 2                   ! Transparent-Modus   !
   If Left$(Dum$(M.i%+Pm1%))="-" ! 1.Zeichen =  -  ?  |- Menü
     Deftext ,2                  ! DEFTEXT grau        |  zeichnen
     Text Mxl2%+(20/M.xt%),Myo2%+3/M.yt%+M.i%*18/ ...  !
       ... M.yt%,100/M.xt%,Right$(Dum$(M.i%+Pm1%), ... !
                           ... Len(Dum$(M.i%+Pm1%))-1) !
   Else                          ! Aktive Zeile !      |
     Deftext ,0                  ! DEFTEXT normal      |
     Text Mxl2%+(20/M.xt%),Myo2%+3/M.yt%+M.i%*18/ ...  !
         ... M.yt%-1+M.yt%,100/M.xt%,Dum$(M.i%+Pm1%)   !
   Endif                                               !
 Next M.i%                       ! Nächste Zeile -------'
 Defmouse 3                      ! DEFMOUSE Zeigefinger
 Deffill 1,1,1                   ! DEFFILL schwarz
 Graphmode 3                     ! XOR-Modus
 Boundary 0                      ! V3.0 : P-Rahmen aus
 ' Dpoke Vdibase+34,0            ! V2.xx: P-Rahmen aus
 Repeat                          ! Auswahl-Schleife >----------.
   On menu                       ! Ereignis-Überwachung        |
   Mouse Xko.x%,Yko.y%,Msk%      ! Maus-Status holen           |
   Yi%=Int((Yko.y%-(Myo2%-(8/M.yt%)))/(18/M.yt%))              !
   '                             ! Zeilen-Index berechnen      |
   If Xko.x%>Mxl2%+(12/M.xt%) And Yi%>0 And Xko.x%< ...        !
                ... Mxr2%-(13/M.xt%) And Yi%<=(Mmx%)           !
                                 ! Maus auf einem Menüpunkt?   |
     If Left$(Dum$(Yi%+Pm1%))<>"-"  ! Menüpunkt aktiv?         |
       Pbox Mxl2%+(14/M.xt%),Myo2%-(5/M.yt%)+Yi%*... ! in-     |
       ... (18/M.yt%)-1+M.yt%,Mxr2%-(14/M.xt%)+1-... ! ver-    |
           ... M.xt%,Myo2%+(5/M.yt%)+Yi%*(18/M.yt%) ! tie-     |
     Endif                                          ! ren      |
     Repeat                      ! Mausbewegung abwarten >---. |
       Yi2%=Int((Mousey-(Myo2%-(8/M.yt%)))/(18/M.yt%))       | |
       '                         ! Ggfs. neuen Index holen   | |
       Msk%=Mousek               ! Maustasten-Status holen   | |
       M.key$=Inkey$             ! Tastatur abfragen         | |
       On menu                   ! Ereignis-Überwachung      | |
     Until Mousex<Mxl2%+(12/M.xt%) Or Mousex>Mxr2%- ...      | |
     ... (13/M.xt%) Or Yi%<>Yi2% Or Msk%>0 Or M.key$>"" <-' |
     If Left$(Dum$(Yi%+Pm1%))<>"-"  ! Menüpunkt aktiv?         |
       Pbox Mxl2%+(14/M.xt%),Myo2%-(5/M.yt%)+Yi%*... ! In-     |
       ... (18/M.yt%)-1+M.yt%,Mxr2%-(14/M.xt%)+1-... ! ver-    |
           ... M.xt%,Myo2%+(5/M.yt%)+Yi%*(18/M.yt%) ! tie-     |
     Else                        ! Menüpunkt ist inaktiv!      |
       Clr Msk%                  ! Maustasten-Status löschen   |
     Endif                                                     |
   Else                          ! Maus nicht auf Menüpunkt!   |
     Clr Yi2%                    ! Punkt-Index löschen         |
   Endif                                                       |
   Exit if (Mousex<Mxl% Or Mousex>Mxr% Or ...                  |
   ... Mousey<Myo% Or Mousey>Myu%) And Yi2%<=Mmx%              |
   ' Schleife verlassen, wenn sich der Mauszeiger außerhalb    |
   ' des Darstellungsbereichs befindet.                        |
   M.key$=Inkey$                 ! Tastatur abfragen           |
 Until (Msk%>0 Or M.key$>"") And Yi2%<=Mmx%                    |
 '   Schleife verlassen, wenn Tastatur betätigt oder ein       |
 '   Mausknopf gedrückt wurde.! <------------------------------'
 Defmouse 0                      ! DEFMOUSE Pfeil
 Deffill 1,0,0                   ! DEFFILL weiß
```

```
  Graphmode 1                    ! Replace-Modus
  Put Max(0,Mxl2%),Max(0,Myo2%),Mmen$ ! Hintergrund restaurieren
  Boundary 1                     ! V3.0 : P-Rahmen an
  ' Dpoke Vdibase+34,1           ! V2.xx: P-Rahmen an
  Swap *F.adr%,Dum$()            ! Menütextfeld wieder global
  Erase Dum$()                   ! Lokales Swap-Feld löschen
  *V.adr%=Yi2%                   ! Gewählten Index zurückgeben
  Pause 5                        ! Kleine Klickpause
Return
```

Weitere Beispiele zu GET finden Sie hier im Buch in Hülle und Fülle.

HIDEM { HI } — Mauszeiger ausschalten

HIDEM

Der Mauszeiger wird ausgeschaltet, die Koordinatenermittlung kann jedoch weiter durch MOUSE-Befehle durchgeführt werden. Bei FILESELECT- und ALERT-Aufrufen wird intern immer SHOWM ausgeführt. Nach Box-Bedienung wird intern automatisch wieder HIDEM ausgeführt, sofern vor Box-Aufruf HIDEM gültig war.

Beispiele hierzu finden Sie unter anderem unter DATA in der Prozedur Pcode unter CALL in der Prozedur Scroll, unter PSET in der Prozedur Raster und im Beispiel zu SHOWM.

PUT { PU } — Bildschirmbereich setzen

PUT X_links,Y_oben,Var$ [,Modus]

PUT (nicht zu verwechseln mit dem Diskettenbefehl PUT#) zeichnet einen durch GET (siehe dort) eingelesenen Bildausschnitt an die Koordinaten X_links/Y_oben. Die bei GET definierte Größe bleibt dabei unverändert. Die Koordinaten des unteren rechten Eckpunktes ergeben aus der Breite und Höhe des mit GET gespeicherten Ausschnitts. Dabei kann die Position auch so gewählt werden, daß der Bildausschnitt teilweise oder auch völlig außerhalb des Bildschirmbereichs liegt.

Durch die Option Modus kann ein Grafikmodus bestimmt werden, sonst wird im Replace-Modus gezeichnet. Negative Werte für Modus sollten tunlichst vermieden werden, da sonst mit Fehlfunktionen zu rechnen ist.

Modus:

0 = Ziel = 0
Der Zielbereich wird gelöscht.

1 = Ziel = Quelle AND Ziel
Nur die Bildschirmpunkte werden im Zielbereich gezeichnet, die vorher in Quelle UND Ziel gesetzt waren.

2 = Ziel = Quelle AND (NOT Ziel)
Im Zielbereich werden nur dort Punkte gezeichnet, wo vorher in Quelle Punkte gesetzt UND in Ziel NICHT gesetzt waren.

3 = Ziel = Quelle *Entspricht GRAPHMODE 1*
Quelle wird komplett in Ziel übertragen.

4 = Ziel = (NOT Quelle) AND Ziel
Im Ziel werden nur die Punkte gesetzt, die vorher in Quelle NICHT gesetzt UND in Ziel gesetzt waren.

5 = Ziel = Ziel

6 = Ziel = Quelle XOR Ziel *Entspricht GRAPHMODE 3*
Es werden im Zielbereich nur dort Punkte gesetzt, wo vorher in Quelle ODER Ziel ABER NICHT in beiden zugleich Punkte gesetzt waren.

7 = Ziel = Quelle OR Ziel *Entspricht GRAPHMODE 2*
In Ziel werden Punkte gesetzt, wenn sie vorher in Quelle ODER in Ziel gesetzt waren.

8 = Ziel = NOT (Quelle OR Ziel)
Im Ziel werden nur die Punkte gesetzt, die vorher weder in Quelle noch in Ziel gesetzt waren.

9 = Ziel = NOT (Quelle XOR Ziel)
Quelle und Ziel werden XOR-verknüpft (siehe Modus 6) und dann im Ziel nur die Punkte gesetzt, die in dem XOR-Ergebnis NICHT gesetzt sind.

10 = Ziel = NOT Ziel
Im Ziel werden die Punkte gesetzt, die vorher in Ziel NICHT gesetzt waren (Ziel wird invertiert).

11 = Ziel = Quelle OR (NOT Ziel)
Ziel wird vor Quell-Verknüpfung invertiert. Es werden dann nur die Zielpunkte gesetzt, die in Quelle ODER im invertierten Ziel gesetzt sind.

12 = Ziel = NOT Quelle
Es werden im Ziel nur die Punkte gesetzt, die in Quelle NICHT gesetzt waren.

13 = Ziel = (NOT Quelle) OR Ziel   Entspricht GRAPHMODE 4
Quelle wird vor Ziel-Verknüpfung invertiert. Es werden dann nur die Zielpunkte gesetzt, die in der invertierten Quelle ODER in Ziel gesetzt sind (revers transparent).

14 = Ziel = NOT (Quelle AND Ziel)
Nach einer AND-Verknüpfung von Quelle und Ziel werden anschließend nur die Punkte im Ziel gesetzt, die im Verknüpfungsergebnis NICHT gesetzt sind.

15 = Ziel = 1
Im Ziel werden alle Punkte gesetzt.

Das folgende Beispiel stellt die Verknüpfungsmodi im Monochrom-Modus dar. Im Farbbetrieb ist das Verknüpfungsergebnis von den Farbwerten der beteiligten Farbregister abhängig und nur sehr schwer voraussagbar.

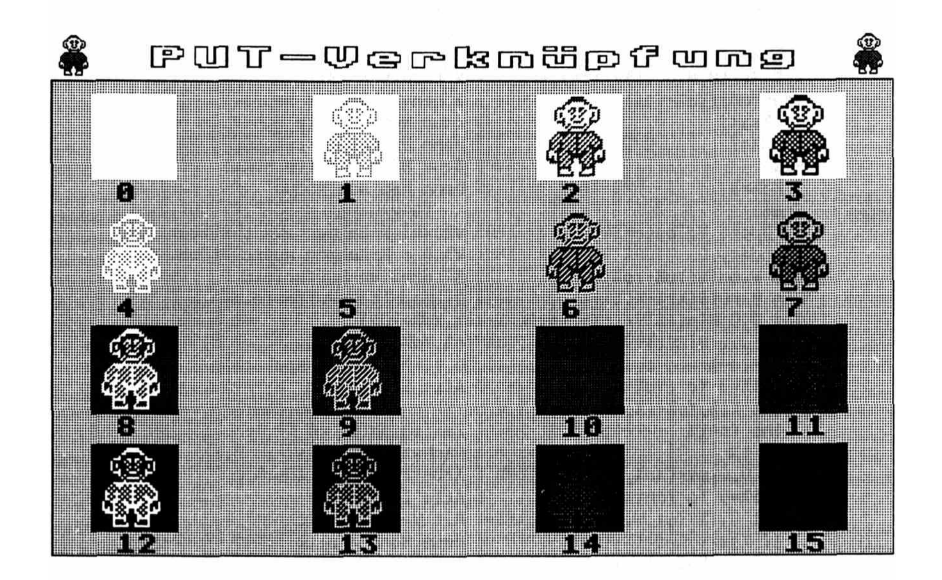
Im Beispiel wird eine Routine verwendet (Size), die die Aufgabe hat, einen beliebigen Bildschirmausschnitt in beliebiger Richtung zu vergrößern oder zu verkleinern. Dabei wird vorübergehend die linke obere Bildschirmecke zur Produktion des neuen Bildes "mißbraucht", jedoch nach Erledigung wieder restauriert.

Beispiel (nur Hires):

```
For I%=1 To 67                  ! 3 Header-Words + 64 Daten-Words
  Read A%                       ! DATA lesen
  A$=A$+Mki$(A%)                ! PUT-String bilden
Next I%                         ! Nächstes Word
Deftext ,16,,12                 ! DEFTEXT outlined
Text 100,25,"PUT-Verknüpfung"
@Size(100,2,400,30,540,28,*B$) ! Text verlängern
Put 80,2,B$                     ! Text-Image zeichnen
Deftext ,0,,12                  ! DEFTEXT normal
Put 8,2,A$                      ! Männchen steht...
Put 600,2,A$                    ! ... Spalier
@Size(8,2,40,34,64,64,*A$)      ! Männchen vergrößern
Print At(10,10);"Bild mit Maus bewegen! (Ende durch Mausklick)"
Repeat                          ! Männchen bewegen   >--.
  Mouse Mx%,My%,Mk%                                     !
  Put Mx%,My%,A$,6              ! Image im XOR-Modus    !
  Put Mx%,My%,A$,6              ! Zweimal auf dieselbe  !
  '                             ! Position.             !
Until Mk%                       ! <---------------------'
Deffill ,2,2                    ! DEFFILL hellgrau
Pbox 6,36,633,393               ! Hintergrundbox zeichnen
Graphmode 2                     ! Transparent-Modus
For I%=0 To 3                   ! 4 mal ...
  For J%=0 To 3                 !    ... 4
    Put 36+I%*166,46+J%*88,A$,J%*4+I% ! Männchen zeichnen
    Text 36+I%*166+18,46+J%*88+80,Str$(J%*4+I%) ! Modus anzeigen
  Next J%
Next I%
' Männchen-DATAs
Data 31,31,1,0,0,7,49152,15,57344,24,12288,32,2048,230
Data 52736,266,41216,324,17664,322,34048,292,18688,227
Data 36352,40,10240,23,53248,8,8192,255,64512,373,22016
Data 683,43776,1365,21888,1771,43648,1365,23936,1771,44928
Data 2525,30272,2223,60480,2775,54592,3756,44480,2004,55168
Data 168,27648,248,31744,136,17408,132,33792,252,64512,0,0
'
Procedure Size(Xl%,Yo%,Xr%,Yu%,Xd%,Yd%,Buf%)
  ' Für Hires/Midres/Lowres !
  ' Streckt oder staucht einen beliebigen Bildschirmausschnitt
  ' in X- und/oder Y-Richtung und liefert das Ergebnis in einem
  ' PUT-Rückgabe-String.
  '
  '  Xl% = Linke Quell-X-Koordinate
  '  Yo% = Obere Quell-Y-Koordinate
  '  Xr% = Rechte Quell-X-Koordinate
  '  Yu% = Untere Quell-Y-Koordinate
  '  Xd% = Breite des neuen Bildes in Pixel
  '  Yd% = Höhe des neuen Bildes in Pixel
```

```
' Buf% = Pointer auf eine Rückgabe-String-Variable, die nach
'        Rückkehr den erzeugten Ausschnitt enthält. Dieser
'        kann anschließend durch PUT an eine beliebige
'        Bildschirmposition gesetzt werden.
Local Bg$,Xsz,Ysz,I%,J%,Xcnt%,Ycnt%,Md%,Xt%,Yt%,Buf$
Yt%=Min(2,3-Xbios(4))            ! Y-Auflösungsteiler
Xt%=2-Sgn(Xbios(4))              ! X-Auflösungsteiler
Graphmode 1                      ! Replace-Modus
Deffill ,0,0                     ! DEFFILL weiß
Yu%=Max(0,Min((399/Yt%),Yu%)) ! Y_unten  -. innerhalb des
Xr%=Max(0,Min((639/Xt%),Xr%)) ! X_rechts -' Bildschirms halten
Ysz=Yd%/(Yu%-Yo%)                ! Höhenverhältnis neu/alt
If Ysz<1                         ! Neue Höhe < alte Höhe?
  Md%=7                          ! Dann Transparent-Modus
Else                             ! Neue Höhe => alte Höhe!
  Md%=3                          ! Dann Replace-Modus
Endif
'  Dpoke Vdibase+34,0            ! V2.xx: P-Rahmen ausschalten
Boundary 0                       ! V3.0 : P-Rahmen ausschalten
Dim V.buf$(Yu%-Yo%)              ! Bildzeilen-Pufferfeld
Dim H.buf$(Xr%-Xl%)              ! Bildspalten-Pufferfeld
Get 0,0,Min((639/Xt%),Max(Xr%-Xl%,Xd%+10)), ...
    ... Min((399/Yt%),Max(Yu%-Yo%,Yd%+10)),Bg$
' Hintergrund des Arbeitsbereichs (links oben) speichern
For I%=0 To (Yu%-Yo%)            ! Alle Bildzeilen  >------.
  Get Xl%,Yo%+I%,Xr%,Yo%+I%,V.buf$(Ycnt%)  ! speichern      !
  Inc Ycnt%                      ! Zeilenzähler +1         !
Next I%                          ! Nächste Zeile  <--------'
Pbox 0,0,max(Xd%,Xr%-Xl%),Yd% ! Arbeitsbereich löschen
For I%=0 To Ycnt%-1              ! Alle Zeilen  >---------.
  Put 0,I%*Ysz,V.buf$(I%),Md% ! Mit Versatz zeichnen    !
  If Ysz>1 And I%<Ycnt%-1        ! Neue Höhe > alte Höhe?  !
    For J%=0 To Ysz              ! Dann Ysz Zeilen...      !
      Put 0,I%*Ysz+J%,V.buf$(I%),Md%   ! ... auffüllen     !
    Next J%                                                !
  Endif                                                    !
Next I%                          ! Nächste Zeile <---------'
Xsz=Xd%/(Xr%-Xl%)                ! Breitenverhältnis neu/alt
If Xsz<1                         ! Neue Breite < alte Breite?
  Md%=7                          ! Dann Transparent-Modus
Else                             ! Neue Höhe => alte Höhe!
  Md%=3                          ! Dann Replace-Modus
Endif
For I%=0 To (Xr%-Xl%)            ! Alle Bildspalten  >-----.
  Get I%,0,I%,Yd%,H.buf$(Xcnt%) ! Speichern               !
  Inc Xcnt%                      ! Spaltenzähler +1        !
Next I%                          ! Nächste Spalte  <-------'
Pbox 0,0,Xd%,Yd%                 ! Arbeitsbereich löschen
For I%=0 To Xcnt%-1              ! Alle Spalten  >-----------.
  Put I%*Xsz,0,H.buf$(I%),Md% ! Mit Versatz zeichnen         !
  If Xsz>1 And I%<Xcnt%-1        ! Neue Breite > alte Breite?  !
    For J%=0 To Xsz              ! Dann Xsz Zeilen...          !
      Put I%*Xsz+J%,0,H.buf$(I%),Md%   ! ... auffüllen         !
    Next J%                                                    !
  Endif                                                        !
Next I%                          ! Nächste Spalte  <-----------'
Get 0,0,Min((639/Xt%),Xd%),Min((399/Yt%),Yd%),Buf$
'                                ! Neues Bild speichern
Put 0,0,Bg$                      ! Arbeitsbereich wieder räumen
```

```
    Erase V.buf$()                  ! Spalten-Puffer löschen
    Erase H.buf$()                  ! Zeilen-Puffer löschen
    ' Dpoke Vdibase+34,1            ! V2.xx: P-Rahmen einschalten
    Boundary 1                      ! V3.0 : P-Rahmen einschalten
    *Buf%=Buf$                      ! Neuen PUT-String zurückgeben
  Return
```



9.5.1 Organisation eines PUT-Strings

In diesem Zusammenhang ist es vielleicht angebracht, ein paar Worte zur PUT-String-Organisation zu verlieren.

Die ersten 3 Words (6 Bytes) des Strings enthalten der Reihe nach:

Word 1
Die Breite des Ausschnitts minus 1 (= X_rechts minus X_links).

Word 2
Die Höhe des Ausschnitts minus 1 (= Y_unten minus Y_oben).

Word 3
Anzahl der Bit-Planes (Hires = 1/Midres = 2/Lowres = 4).

In Hires entspricht ein gesetztes Bit im PUT-Raster einem gesetzten Punkt im gespeicherten Bild-Ausschnitt.

An diesen sogenannten "Header" (Kopf) werden der Reihe nach die Bit-Informationen des gespeicherten Ausschnitts angehängt.

Dies geschieht in Hires nach folgendem Schema: Angenommen, der Ausschnitt ist 26 Pixel breit und 15 Pixel hoch. Die Breite des Auschnitts wird durch 16 (Wordbreite) geteilt und - falls bei der Teilung ein Rest geblieben ist - zum integrierten Teilungsergebnis der Wert 1 hinzugezählt:

```
Word 1  =  Int(26/16)+Abs((26 MOD 16)>0)
```

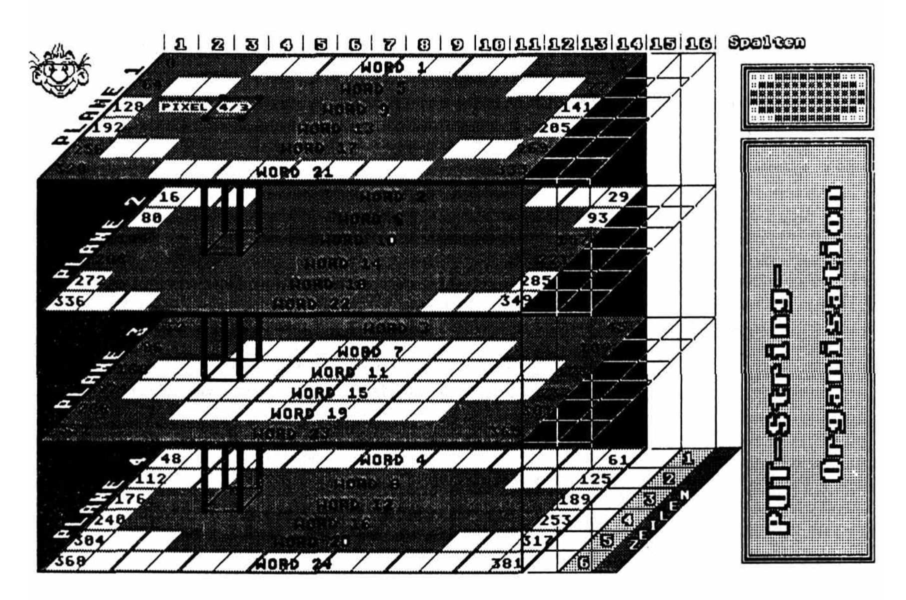
Die Bit-Raster-Breite im PUT-String ist also immer glatt durch 16 teilbar. Im obigen Beispiel belegen die Pixel-Bits 17-26 der ersten Ausschnittzeile die Bits 0 bis 9 des 5. String-Words, während die verbleibenden Bits 10 bis 15 des 5. Words mit "irgend etwas" gefüllt werden. Diese nicht benötigten Bits enthalten also unbrauchbaren Bit-Müll.

```
|<-------------------  32 Bit  ( = 2 Words )  --------------->|
|<-------  Ausschnittbreite = 27 Pixel  --------->|<- 'Müll'->|
|                                                 |           |
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 0 1 0 0   1
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 1 0 0 0 0   2
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 0 1 0 0   3
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 0 0 1 0   4
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 1 0 0 0   5
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 1 1 0 0   6
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 0 0 0 0   7
* * * * * * * *   B I L D - D A T E N   * * * * * * 0 0 0 0 0 0   8
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 1 0 1 0 0   9
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 0 0 0 0 0  10
* * * * * * * * * * * * * * * *|* * * * * * * * * * 1 0 0 0 1 0  11
* * * * * * * * * * * * * * * *|* * * * * * * * * * 1 0 0 0 0 0  12
* * * * * * * * * * * * * * * *|* * * * * * * * * * 1 0 1 0 1 1  13
* * * * * * * * * * * * * * * *|* * * * * * * * * * 0 1 1 1 0 0  14
* * * * * * * * * * * * * * * *|* * * * * * * * * * 1 1 0 0 1 0  15
|<--vorderes Word der Zeile--->|<--hinteres Word der Zeile--->|
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F
(Hex-Bitnummern)
```

In diesem Rythmus werden nun alle Bildzeilen im Ziel-String abgelegt. Bit 0 - 15 des 6. Words und Bit 0 - 9 des 7. Words enthalten die Daten der 2. Pixel-Zeile, Bit 0 - 15 des 8. Words und Bit 0 - 9 des 9. Words enthalten die Daten der 3. Pixel-Zeile usw.

In Midres oder Lowres wird das Verfahren wesentlich komplizierter. Ich habe fast einen ganzen Tag damit verbracht, mir zu überlegen, wie

ich dieses Verfahren transparent machen könnte. Und zwar so, daß es für jeden leicht verständlich ist. Nach vielen erfolglosen Ansätzen habe ich mich entschlossen, die vielen Seiten Text, die ich dabei verschwenden würde, durch eine möglichst aussagekräftige Grafik zu ersetzen und diese dann zu erklären.



Prinzipiell ist es in Farbe das gleiche Schema, nur daß sich die Farbinformationen eines Bildschirmpunktes daraus ergeben, daß 2 (Midres) bzw. - wie in der Grafik dargestellt - 4 (Lowres) Bitraster (Planes) im oben beschriebenen Format "übereinander" gelegt werden. Das für den jeweiligen Punkt zuständige Farbregister ergibt sich aus der Bit-Summe der jeweiligen Plane-Bits. Grundsätzlich ist ein Farb-GET/PUT-Image genauso organisiert wie der Farb-Bildschirm, wobei nur jeweils die nicht benötigten Rest-Bits (Bit-Müll) des letzten Words einer Zeile zu beachten sind.

In der Grafik habe ich die Bits der Reihenfolge nach durchnummeriert. Als Beispiel habe ich ein 14*6-Raster gewählt. Nehmen Sie an, Sie "schneiden" mittels GET aus dem Bildschirm ein Teil mit der Breite von 14 und der Höhe von 6 Pixel aus. Die ersten sechs Bytes des String-Headers habe ich ja oben schon erklärt. Daran schließen sich nun - im Beispiel - im 4-Word-Rhythmus die Bit-Informationen der einzelnen Zeilen an.

Legt man nun die vier Planes direkt übereinander, so ergibt die Bit-Stellung der vier zusammengehörigen Bits die Nummer des Farbregisters, aus dem dieser Bildschirmpunkt seine Farbe bezieht. In der Grafik habe ich das Ausschnitt-Pixel 4/3 besonders herausgestellt, um daran die Farbberechnung zu demonstrieren.

```
In Plane 1 ist dieses Bit gesetzt     :  2^0  =  1
In Plane 2 ist es ebenfalls gesetzt  :  2^1  =  2
In Plane 3 ist es nicht gesetzt       :    0  =  0
In Plane 4 ist es wiederum gesetzt   :  2^3  =  8
                                         -----------
Die Summe dieser vier Bits ergibt     :         11
                                         ===========
```

Das Pixel 4/3 des Ausschnitts bezieht seine Farbe also aus dem Farbregister 11. Zur Vertiefung nehme ich nun noch die vier Bits 205, 221, 237 und 317 am rechten Rand.

```
In Plane 1 ist das Bit nicht gesetzt :    0  =  0
In Plane 2 ist es gesetzt             :  2^1  =  2
In Plane 3 ist es ebenfalls gesetzt  :  2^2  =  4
In Plane 4 ist es nicht gesetzt       :    0  =  0
                                         -----------
            Für Pixel 14/4 ist das Farbregister 6 zuständig.
                                         ===========
```

In der Grafik sehen Sie, daß die beiden letzten Bits jeder Zeile leer sind. An Speicherplatz wird jedoch für jedes "angebrochene" Word ein volles Word benötigt. Wäre also in diesem Fall der Ausschnitt 17 Punkte breit, so würde dafür der doppelte Speicherplatz erforderlich, obwohl vom zweiten Word jeder Zeile nur das 1 Bit belegt wäre.

Anhand der Word-Numerierung können Sie erkennen, in welcher Folge die Words im Speicher liegen.

Als Beweis für die Richtigkeit dessen, was ich gesagt habe, folgt nun ein kleines Listing (gilt nur für Lowres). Dieses Listing ist hier durch die eingestreuten Kommentare und Erläuterungen etwas in die Länge gezogen. Gehen Sie davon aus, daß ab hier alle Listing-Zeilen bis zur abschließenden Prozedur Gplane zusammengehören.

```
Dim Tos%(15),Vdi%(15)          ! TOS- und VDI-Tabellen
For I%=0 To 15                 ! 15 Farbregister
  Read Tos%(I%)                ! TOS-Index lesen
Next I%
For I%=0 To 15                 ! 15 Farbregister
  Read Vdi%(I%)                ! VDI-Index lesen
Next I%
' Umrechnungstabelle VDI -> TOS
Data 0,15,1,2,4,6,3,5,7,8,9,10,12,14,11,13
```

```
' Umrechnungstabelle TOS -> VDI
Data 0,2,3,6,4,7,5,8,9,10,11,14,12,15,13,1
```

Zuerst brauchen wir dazu zwei Umrechnungstabellen, die uns die Möglichkeit geben, die verwendeten Farben auch nachprüfen zu können. Wenn man nämlich über einen VDI-Befehl (COLOR, DEFFILL, DEFTEXT etc.) eine Farbe bestimmt und dann eine Figur damit zeichnet, wird intern nicht der VDI-Index zur Farbbestimmung verwendet, sondern dieser wird intern vor Befehlsausführung in den TOS-(Hardware-)Index umgerechnet. Wird nun wiederum die Farbe eines Bildschirmpunktes über die VDI-Funktion POINT() gelesen, wird auch hier vor der Wertrückgabe der Farbwert vom TOS- in den VDI-Index zurückgewandelt. Die Line-A-Funktion PTST() liest dagegen den Bildschirmpunkt im TOS-Index. Übrigens bedienen sich alle Line-A-Befehle des TOS-Index'.

Ehrlich, Sie haben mein tiefes Mitgefühl bei dem verzweifelten Versuch, dieses Mischmasch zu verstehen (was ist denn VDI?, warum TOS?, wieso überhaupt?), aber vor dem Erfolg steht nun einmal der Schweiß.

Es kommt noch besser. Es bleibt nämlich nicht dabei, sich die näheren Zusammenhänge vor das "geistige Auge" zu halten, sondern diese auch aus den geistigen Tiefen in die Realität eines beweisbaren Algorithmus zu zerren. Aber zusammen schaffen wir das.

```
Deffill 2,2,8               ! VDI-Register 2 für Füllmuster
Pbox 295,5,318,21           ! Kleine Hintergrund-Box zeichnen
For I%=0 To 5               ! -------------.
  For J%=0 To 13                           ! Grafik vorbereiten
    Read A$                                ! (die Datas enthalten
    A%=Val("&H"+A$)         !   TOS-Farbe  |- im Hexa-Format die TOS-
    Color Vdi%(A%)          !-> VDI Index  |  Farbwerte für die
    Plot 300+J%,10+I%       ! Punkt setzen !  einzelnen Bildpunkte)
  Next J%                                  !
Next I%                     ! -------------'
Data 5,5,5,6,6,6,6,6,6,6,6,5,5,5
Data 5,6,6,B,B,B,B,B,B,B,B,6,6,5
Data 6,B,B,B,B,B,B,B,B,B,B,B,B,6
Data 6,B,B,B,B,B,B,B,B,B,B,B,B,6
Data 5,6,6,B,B,B,B,B,B,B,B,6,6,5
Data 5,5,5,6,6,6,6,6,6,6,6,5,5,5
'
Get 300,10,313,15,A$        ! Erzeugte Grafik speichern
Adr%=Varptr(A$)             ! Startadresse des GET/PUT-Strings
```

Noch einmal: Gleich, wie eine Grafik entsteht - ob die Farben über den TOS- oder den VDI-Index - ob Line-A- oder VDI-Grafikbefehle

verwendet wurden - der durch GET gespeicherte Ausschnitt liegt immer im TOS-Format im Speicher!!

Wenn Sie die Farben durch den VDI-Index bestimmt haben, erwarten Sie bitte nicht, daß die zusammengehörigen 4 Bits eines Bildpunktes im Speicher den VDI-Index wiedergeben. Nehmen wir als Beispiel den in den oben stehenden Datas verwendeten TOS-Farbwert B (11). Die Stellung der 4 zusammengehörigen Plane-Bits im Speicher für einen Punkt dieser Farbe wäre:

```
1 0 1 1          (2^3+2^1+2^0 = 11)
```

Der Index wurde oben durch COLOR in den VDI-Index umgerechnet:

```
1 1 1 0          (2^3+2^2+2^1 = 14)
```

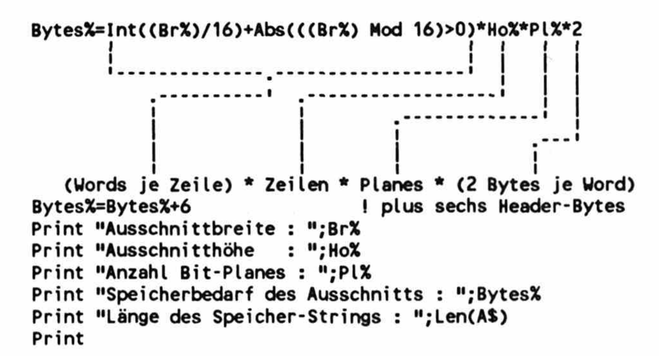
Durch die folgenden 4 Programmzeilen werden nun die vier Bits ermittelt, in den der Plane-Stellung entsprechenden Wert umgerechnet und addiert. Aus dieser Addition ergibt sich wieder der Wert 11 (Hex: B).

```
Bit4.3%=2^0*Abs((Dpeek(Adr%+6+16) And 2^12)>0)
'      Plane-1-Bit = Bit 12 (4.Bit v.links) des 9.Words
'      hinter dem String-Header (= 12. Word des Strings)
Bit4.3%=Bit4.3%+2^1*Abs((Dpeek(Adr%+6+18) And 2^12)>0)
'      Plane-2-Bit = Bit 12 des 10.Words hinter dem Header
Bit4.3%=Bit4.3%+2^2*Abs((Dpeek(Adr%+6+20) And 2^12)>0)
'      Plane-3-Bit = Bit 12 des 11.Words hinter dem Header
Bit4.3%=Bit4.3%+2^3*Abs((Dpeek(Adr%+6+22) And 2^12)>0)
'      Plane-4-Bit = Bit 12 des 12.Words hinter dem Header
'
Print "Farbwert für Ausschnitt-Pixel 4/3 : ";Bit4.3%
```

Die Berechnungszeilen für den Farbwert sind vielleicht etwas schwer zu durchschauen. Zuerst erzeuge ich in der Zeile den Bitwert, der der Stellung des Bits in der Plane-Hierarchie entsprechen würde (2^0, 2^1 etc.). Dieser Wert wird dann mit 1 multipliziert, falls das Bit 12 (4. Bit von links) des betreffenden Words gesetzt ist, anderenfalls mit 0. So verfahre ich mit allen vier Plane-Bits und addiere die Werte.

Die folgenden vier Zeilen ermitteln die Breite und Höhe des Ausschnitts, die Anzahl der bei der GET-Speicherung vorhandenen Bit-Planes sowie daraus dann anschließend den Speicherbedarf des GET/PUT-Ausschnitts in Bytes.

```
Br%=Dpeek(Adr%)+1           ! Breite des Auschnitts (hier: 14)
Ho%=Dpeek(Adr%+2)+1         ! Höhe des Ausschnitts  (hier:  6)
Pl%=Dpeek(Adr%+4)           ! Anzahl der Bit-Planes (hier:  4)
```



```
Bytes%=Int((Br%)/16)+Abs(((Br%) Mod 16)>0)*Ho%*Pl%*2

   (Words je Zeile) * Zeilen * Planes * (2 Bytes je Word)
Bytes%=Bytes%+6                 ! plus sechs Header-Bytes
Print "Ausschnittbreite : ";Br%
Print "Ausschnitthöhe   : ";Ho%
Print "Anzahl Bit-Planes : ";Pl%
Print "Speicherbedarf des Ausschnitts : ";Bytes%
Print "Länge des Speicher-Strings : ";Len(A$)
Print
```

Ein weiterer Beweis für die Richtigkeit der Aussagen wird nun durch den nächsten kleinen Programmblock geliefert. Dabei wird das Bildchen Punkt für Punkt mit der (VDI-)Funktion POINT() auf die enthaltenen Farbwerte getestet. Als Ergebnis sehen Sie dieselben Hex-Farbwerte, die eingangs auch in den Datas verwendet wurden.

Hier kommt übrigens die andere Umwandlungstabelle in's Spiel. POINT() liefert die Farben der Bildpunkte im VDI-Index. Um einen Vergleich anstellen zu können, wird dieser Index in der PRINT-Zeile in den TOS-Index umgewandelt.

```
For I%=0 To Ho%-1               ! 6 Zeilen
  For J%=0 To Br%-1             ! 14 Spalten
    Vdi_farbe%=Point(300+J%,10+I%) ! VDI-Farbwerte der Bildpunkte
      Print Right$("0"+Hex$(Tos%(Vdi_farbe%)))' ! als TOS-Index im
  Next J%                                       ! Hex-For. ausgeben
  Print
Next I%
```

Soweit, so gut. Das haben Sie ja alles ganz tapfer durchgestanden. Aber was soll man mit diesem konfusen DPEEK-, POINT()-, COLOR-, GET-Wirrwarr anfangen? Als Belohnung und krönender Abschluß folgt nun eine Prozedur, die Sie aller Sorgen entledigt.

Zuerst speichern wir die kleine Grafik ab:

```
Bsave "BILD.DAT",Varptr(A$),Len(A$)
```

Nachdem Sie das getan haben, wechseln Sie auf den Monochrom-Monitor und starten das folgende Program (mitsamt der Prozedur Gplane natürlich).

Sie haben vielleicht keinen Hires-Monitor? Das macht nichts. Die Routine funktioniert auch für Lowres->Midres-Konvertierungen. Die

hier gemachten Angaben beziehen sich jedoch hauptsächlich auf Lowres->Hires-Konvertierungen, was wohl auch das vorwiegende Einsatzgebiet der Prozedur ist.

```
Dim Feld!(1)                  ! Dummy-Swap-Feld einrichten
Open "i",#1,"BILD.DAT"        ! Bilddatei öffnen...
A$=Input$(Lof(#1),#1)         ! ...und einlesen...
Close #1                      ! ...und wieder schließen
@Gplane(A$,0) ! Aufruf, wenn das Bild komplett dargestellt
'               werden soll. Die Planes werden überlagert.
Print At(1,5);"4 Planes übereinander (<Taste>=Weiter)"
Void Inp(2)
Cls
@Gplane(A$,1) ! Aufruf, wenn die Planes einzeln dargestellt
'               werden sollen.
Print At(1,5);"4 Planes separat (<Taste>=Weiter)"
Void Inp(2)
Cls
@Gplane(A$,*Feld!()) ! Aufruf, wenn die Planes in einem
'                      Bit-Feld zurückgegeben werden sollen.
'
For I%=1 To Dpeek(Varptr(A$)+4)          ! Alle Planes
  For J%=1 To Dpeek(Varptr(A$))+1        ! Alle Spalten
    For K%=1 To Dpeek(Varptr(A$)+2)+1    ! Alle Zeilen
      Color Abs(Feld!(I%,J%,K%))         ! COLOR 0 oder 1
      Plot I%*Br%+2+J%,2+K%              ! Punkt setzen
    Next K%
  Next J%
Next I%
Print At(1,5);"4 Planes separat aus Bit-Feld gelesen"
```

Wie unten in der Prozedur Gplane beschrieben, enthält das Bitfeld Feld!() die Bit-Informationen der einzelnen Planes. Dabei entspricht die Indizierung des Feldes der Lage der einzelnen Lowres- bzw. Midres-Pixel innerhalb des Bildes bzw. Ausschnitts:

```
Feld!(2,87,113)
```

Enthält das Bit des Pixel 87/113 aus der 2. Plane. Alle Dimensionen beginnen dabei mit dem Index 1:

```
Feld!(1,1,1)
```

Enthält das Bit des ersten Pixel (von links) der ersten Zeile (von oben) der ersten Plane. Was Sie nach Rückkehr aus der Prozedur mit diesem Feld machen, ist Ihrer Phantasie überlassen. Auf der beiliegenden Diskette finden Sie übrigens in GPLANE_X.LST noch einmal dieselbe Routine. Diese ist dann allerdings speziell auf V3.0 zugeschnitten und weitestmöglich geschwindigkeits-optimiert.

```
Procedure Gplane(Pstr$,Fld%)
' Konvertiert Lowres- oder Midres-Auschnitte oder
```

```
' -Fullscreens nach Hires, bzw. Lowres-Ausschnitte
' oder -Fullscreens nach Midres.
'
' Pstr$ =  String, der die Bilddaten enthält
'          (entweder GET/PUT-String, Standard-Fullscreen
'          oder Degas-Fullscreen).
' Fld%  =  Entweder Pointer oder Flag.
'          Fld% = 0
'          Flag, das besagt, daß das zu konvertierende Bild
'          komplett dargestellt werden soll. Die einzelnen
'          Bit-Planes werden dabei überlagert. Aus der Stellung
'          der gesetzten Pixel kann abgelesen werden, welche
'          Farbe das ursprüngliche Pixel hatte.
'          Midres:
'                        Plane-Bit
'                        .---.---.
'                        | 0 | 1 |
'                        '---'---'
'          Lowres:
'                        Plane-Bit
'                        .---.---.
'                        | 0 | 1 |
'                        |---+---|
'                        | 2 | 3 |
'                        '---'---'
'          Ist ein Lowres-Pixel in Hires z.B. so aufgeteilt
'
'                          1 0
'                          0 1
'
'          so hatte das entsprechende Pixel in Lowres die
'          Farbe 5 (2^0+2^2).
'
'          Fld% = 1
'          Flag, das besagt, daß von dem Bild alle Planes separat
'          dargestellt werden sollen. Dabei werden die einzelnen
'          Planes nach demselben Schema angezeigt wie bei Fld%=0
'          die einzelnen Pixel.
'
'          Fld% > 1
'          Ist Fld% größer als 1, so wird das als Pointer auf
'          ein Boole-Feld interpretiert, in welchem nach Abschluß
'          die Bit-Informationen der einzelnen Planes abgelegt sind.
'
Local I%,J%,K%,L%,B$,Adr%,Br%,Ho%,Pl%,Words%,Offset%,Ofs%
Local Pl2%,I2%,I3%,I4%,I5%,J2%,J3%,B2$
Adr%=Varptr(Pstr$)          ! Startadresse des Bildes
If Len(Pstr$)=32000 Or Len(Pstr$)=32034
  '                 ! Standard- oder Degas-Fullscreen?
  Br%=320           ! Bzw. Br%=640 für Midres-Bilder (Breite)
  Ho%=200           ! Höhe ist immer 200 Punkte
  Pl%=4             ! Bzw. Pl%=2 für Midres-Bilder (Planes)
  If Len(Pstr$)=32034   ! Degas-Bild?
    Br%=640/(2-Dpeek(Varptr(Pstr$))) ! Dann Breite berechnen
    Pl%=2*(2-Dpeek(Varptr(Pstr$)))   ! Planes des Degas-Bildes
    Ofs%=34                 ! Offset zu den Bilddaten
  Endif
Else                        ! GET/PUT-Ausschnitt
  Br%=Dpeek(Adr%)+1         ! Breite = 1. Word des Strings
```

```
    Ho%=Dpeek(Adr%+2)+1        ! Höhe = 2. Word des Strings
    Pl%=Dpeek(Adr%+4)          ! Planes = 3. Word des Strings
    Ofs%=6                     ! Offset zu den Bilddaten
  Endif
  Pl2%=Pl%*2                   ! Bytes je Plane je Word
  Words%=(Int((Br%)/16)+Abs(((Br%) Mod 16)>0))
  ' Anzahl der Words je Zeile und Plane
  B2$=String$(16,"0")          ! Binär-String-Puffer vorbereiten
  If Fld%>1                    ! Feld-Pointer übergeben?
    Dim Bitfeld!(Pl%,Br%,Ho%)  ! Bit-Feld dimensionieren
  Endif
  For I%=0 To Pl%-1            ! Alle Planes
    I2%=Adr%+Ofs%+I%*2               !---.
    I3%=Int(I%/(Pl%/2))*Ho%          ! Vorbereitung von Konstanten
    I4%=(I% Mod (Pl%/2))*Br%-1       ! zur Zeit-Optimierung
    I5%=(I% Mod (Pl%/2))-1           !---'
    For J%=0 To Ho%-1
      J2%=J%*(Words%*Pl2%)           !---. Vorbereitung von Konstanten
      J3%=J%*Xbios(4)+I%/(Pl%/2)!---' zur Zeit-Optimierung
      Clr B$                         ! Binär-String löschen
      For K%=0 To Words%-1           ! Alle Words der Zeile
        Offset%=I2%+J2%+K%*Pl2% ! Word-Offset zum String-Anfang
        B$=B$+Right$(B2$+Bin$(Dpeek(Offset%)),16) ! Binär-String
        ' Ich bin hier einen etwas ausgefallenen Weg gegangen, um
        ' die - manchmal - gewaltige Bit-Kette zusammenzufügen. Es
        ' können immerhin bis zu 640 Bits in einer Zeile stehen -
        ' und einen 640-Bit-Wert kann noch kein Personal-Computer
        ' verarbeiten. Das wäre eine Zahl mit immerhin 193 Stellen.
      Next K%
      B$=Left$(B$,Br%)               ! Relevante Bit-Kette abschneiden
      For L%=1 To Br%                ! Alle Bits der Kette
        If Fld%>1                    ! Feld-Pointer wurde übergeben
          Bitfeld!(I%+1,L%,J%+1)=(Mid$(B$,+L%,1)="1") ! Dann
          ' jedes Bit in das Feld einsortieren.
        Else                         ! Es wurde kein Pointer übergeben!
          If Mid$(B$,+L%,1)="1" ! L%tes Bit der Kette gesetzt
            If Fld%=0                ! Planes überlagern?
              Plot L%*(Pl%/2)+I5%,J3% ! Pixel zeichnen
            Else                        ! Planes separat!
              Plot L%+I4%,J%+I3%        ! Pixel zeichnen
            Endif
          Endif
        Endif
        Next L%                        ! Nächstes Bit
      Next J%                          ! Nächstes Word
    Next I%                            ! Nächste Plane
    If Fld%>1                          ! Feld-Pointer wurde übergeben?
      Swap *Fld%,Bitfeld!()            ! Gefülltes Bit-Feld zurückgeben
      Erase Bitfeld!()                 ! Dummy-Swap-Feld löschen
    Endif
  Return
```

Nach allem, was ich (bzw. die ST-Erbauer) Ihnen zugemutet habe(n), bleibt nur noch die Frage, wie man Hires-Bilder nach Midres oder Lowres konvertiert. Ich glaube, daß die Menge an Informationen in diesem Abschnitt ausreichen sollten, um Sie mit der Entwicklung einer

entsprechenden Routine zu betrauen. Eigentlich braucht man ja "nur" den ganzen Algorithmus umzukehren - bloß wie?

Ich wünsche Ihnen viel Spaß bei Ihren "Hausaufgaben".

9.5.2 Organisation des Bildschirm-Speichers

Die oben beschriebene Prozedur ist ja auch in der Lage, komplette Lowres- oder Midres-Fullscreens zu konvertieren. Damit wäre der Aufbau eines Bildschirmspeichers eigentlich auch schon erklärt. Trotzdem will ich hier noch einmal gesondert auf die Organisation des Bildschirmspeichers eingehen.

Der Monochrom-Bildschirm wird genaugenommen in ein Feld von 400 Zeilen zu je 80 Byte (= 32000 Bytes) aufgeteilt. Diese Zeilen hängen wie eine lange Kette aneinander. Das erste Byte des Bildschirms (das ist die Adresse, die mit XBIOS(2) erfragt werden kann), enthält die ersten acht Pixel der obersten Bildschirmzeile. Das zweite Byte folgt rechts davon usw. Bis zum 80. Byte, das in der obersten Zeile ganz rechts außen liegt. Das 81. Byte liegt dann wieder auf der linken Bildschirmseite und zwar in der zweiten Zeile von oben. Das 82. Byte folgt dann wieder rechts davon usw.

```
.-------------------------------------------------------------.
|Byte1     Byte2     Byte3 ....... Byte78    Byte79    Byte80  |
|Byte81    Byte82    Byte83 ....... Byte158   Byte159   Byte160 |
|...............................................................|
| ^ ...................... H I R E S ...................... ^ |
| | 400 Zeilen............ Bildschirm- ............400 Zeilen | |
| v ...................... Speicher ....................... v |
|...............................................................|
|Byte31841 Byte31842 Byte31843 .... Byte31918 Byte31919 Byte31920|
|Byte31921 Byte31922 Byte31923 .... Byte31998 Byte31999 Byte32000|
'---------------------------------------------------------------'
```

Auf diese Art setzt sich der Bildschirm mit der Auflösung von 640 Pixel (80 Bytes zu je 8 Bit = 640) in der Breite und 400 Pixel (400 Zeilen) in der Höhe zusammen, bis die Byte-Kette in der rechten, unteren Bildschirmecke angekommen ist.

Mit diesem Wissen ist es kein Problem, den Bildschirmspeicher in einem Stück auf Diskette zu verfrachten.

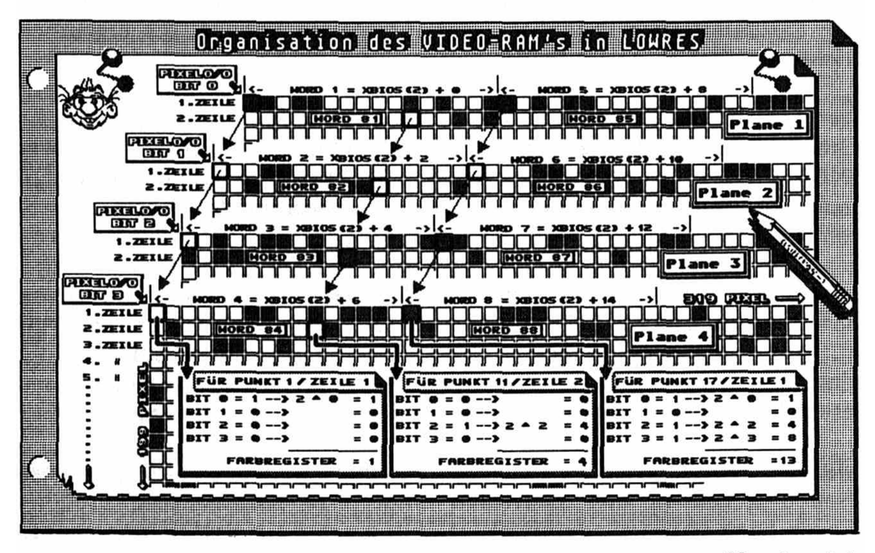
```
BSAVE "SCREEN.DAT",XBIOS(2),32000
```

Eine so gespeicherte Bild-Datei läßt sich auch ebenso einfach wieder in den Bildschirmspeicher zurückladen.

```
BLOAD "SCREEN.DAT",XBIOS(2)
```

Bei Midres- und Lowres-Screens funktioniert das genauso, nur daß die so gespeicherten Screens auf dem Monochrom-Monitor nur sehr verschwommen und schemenhaft erkennbar sind. Das liegt - wie weiter oben schon beschrieben - an den zwei (Midres) bzw. vier (Lowres) untereinander "geklappten" Bit-Planes. Je zwei (Midres) bzw. vier (Lowres) aufeinanderfolgende Words im Speicher bilden die Bit-Informationen für 16 aufeinanderfolgende Bildpunkte. Mit allen weiteren Erläuterungen würde ich mich hier wiederholen.

Ein Bild sagt mehr als tausend Worte. Deshalb folgt noch eine Grafik.



Version 3.0

RC_COPY {RC_} Speicherinternes Rechteck-Copy

RC_COPY Quell,Xq,Yq,Breite,Höhe TO Ziel,Xz,Yz [Modus]

Kopiert das Rechteck mit den Koordinaten Xq/Yq und den Maßen Breite und Höhe aus dem Bildschirm mit der beliebigen Startadresse Quell (Größe = 32000 Byte) an die Koordinaten-Position Xz/Yz des Bildschirms mit der beliebigen Startadresse Ziel (Größe ebenfalls 32000 Byte).

In Modus kann optional ein Verknüpfungsmodus (siehe PUT) angegeben werden. Ohne Angabe von Modus wird im Replace-Modus verknüpft.

Unter "Bildschirm" ist hier nicht unbedingt der reale Bildschirm zu verstehen. Die beiden Speicherbereiche werden lediglich als Bildschirm(-Rechteck) interpretiert. Wenn Sie sich näher mit dem Befehl BITBLT beschäftigt haben, so wird es Ihnen anfangs sicher schwer gefallen sein, diesen Befehl einzusetzen (vielleicht auch immer noch?). Jedenfalls bietet RC_COPY eine sehr wichtige Anwendungsvariante des BITBLTs als BASIC-Befehl. Die Befehle GET und PUT sind ebenfalls BITBLT-Varianten, nur daß diese in ihrem Einsatzbereich auf den logischen Bildschirm (XBIOS(3)) beschränkt sind.

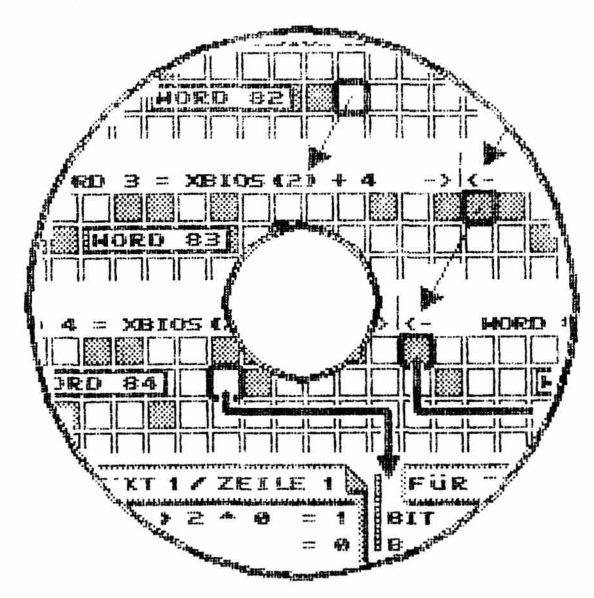
Durch RC_COPY haben Sie die Möglichkeit, beliebige Speicherbereiche im Rechteckmodus miteinander zu verknüpfen. Das heißt also, daß Sie damit auch Daten be- und verarbeiten können, die eigentlich nichts mit Grafik zu tun haben.

Der vorwiegende Einsatzbereich für diesen Befehl bleibt jedoch die Grafik. Er eignet sich z.B. hervorragend dazu, in Verbindung mit dem INLINE-Befehl komplette Image-Bibliotheken zu verwalten. Will man innerhalb eines Programms immer wieder wechselnde Images einsetzen, hat man ohne RC_COPY nur die Möglichkeit, entweder die Images durch BLOAD einzeln in vorbereitete GET/PUT-Strings zu laden oder einen zweiten Bildschirm durch XBIOS(5) zu installieren und von dort aus durch GET/PUT die Images zu transferieren. Oder man lädt mit BLOAD eine Fullscreen in den sichtbaren Bildschirm um dann die einzelnen Images zur späteren Verwendung in ein String-Feld einzulesen. Diese Vorgehensweisen wirken nicht besonders professionell.

Mit RC_COPY lassen sich dagegen beliebige Bildausschnitte aus einer Pseudo-Screen in den sichtbaren Bildschirm holen. Diese Pseudo-Screen kann entweder durch BLOAD in einen 32-KByte-String-Puffer geladen werden oder aber mit INLINE im Programm integriert sein. Von dort aus können Sie nun die Images transferieren, wobei Sie sich nur vorher die Lage und Größe der Images in einem Feld gemerkt haben müssen.

Die zweite wesentliche Anwendungsmöglichkeit ist beim Window-Redraw. Da sich die Größe eines Fensters ständig ändern kann, ist man bei Grafikverwaltung darauf angewiesen, sich im Hintergrund den Inhalt des größtmöglichen Fensters aufzubewahren, um daraus nach Bedarf die nötigen Ausschnitte in das geänderte Fenster zu kopieren.

Bei mehreren, sich überlagernden Windows müssen ggfs. bis zu 4 Teilausschnitte eines unten liegenden Fensters zusammengestellt werden. Dazu werden aus der Rechteckliste des Fensters die Lagen und Maße dieser Rechtecke ausgelesen. Mit RC_COPY ist es dann kein größeres Problem mehr, sich die entsprechenden Fensterteile auf den Bildschirm zu kopieren. Ein Beispiel dazu finden Sie bei den AES-Wind_xxxx-Befehlen.



Beispiel:

```
Print " 32 KByte-Bilddatei wählen"
Fileselect "\*.*","",F$      ! Datei-Auswahl
Cls                          ! clear Screen
If Exist(F$)                 ! Datei existiert?
  Open "i",#1,F$             ! Dann öffnen
  Bild$=Input$(32000,#1)     ! Bild einlesen
  Close #1                   ! Datei schließen
  S%=V:Bild$                 ! Startadresse des Bildes
  D%=Xbios(2)                ! Startadresse des Bildschirms
  For I%=-90 To 269          ! 360 Grad
    X%=150+Sinq(I%)*140      ! X-Ellipsenpunkt
    Y%=100+Cosq(I%)*80       ! Y-Ellipsenpunkt
    Rc_copy S%,I%+90,0,1,50 To D%,X%,Y%      ! Aufruf
    ' @Rc_copy(S%,I%+90,0,1,50,D%,X%,Y%,3)   ! V2.xx-Aufruf
  Next I%                                      (siehe BITBLT)
  Pause 50
  Cls
  For J%=70 To 90 Step 10   ! Radius
    For I%=0 To 359 Step 10 !360 Grad
      X%=150+Sinq(I%)*J%    ! X-Kreispunkt
      Y%=100+Cosq(I%)*J%    ! Y-Kreispunkt
```

```
      Rc_copy S%,X%,Y%,18,18 To D%,X%,Y%    ! Aufruf
      ' @Rc_copy(S%,X%,Y%,18,18,D%,X%,Y%,3) ! V2.xx-Aufruf
    Next I%                                   (siehe BITBLT)
  Next J%
  Pause 50
  Cls
  For I%=0 To 79            ! Was dieses kleine Programm
    For J%=I% To 79-I%      ! produziert, läßt sich nicht
      J2%=J%*8              ! in wenigen Worten erklären.
      I2%=I%*8              ! Schauen Sie es sich an!
      J3%=J%*5
      I3%=I%*5
      ' 4 mal V2.xx-Aufrufe (siehe unter BITBLT)
      ' @Rc_copy(S%,J2%-I2%,J3%+I3%,8,5,D%,J2%-I2%,J3%+I3%,3)
      ' @Rc_copy(S%,J2%+I2%,J3%-I3%,8,5,D%,J2%+I2%,J3%-I3%,3)
      ' @Rc_copy(S%,J2%,J3%-I3%,8,5,D%,J2%,J3%-I3%,3)
      ' @Rc_copy(S%,J2%,J3%+I3%,8,5,D%,J2%,J3%+I3%,3)
      Rc_copy S%,J2%-I2%,J3%+I3%,8,5 To D%,J2%-I2%,J3%+I3%
      Rc_copy S%,J2%+I2%,J3%-I3%,8,5 To D%,J2%+I2%,J3%-I3%
      Rc_copy S%,J2%,J3%-I3%,8,5 To D%,J2%,J3%-I3%
      Rc_copy S%,J2%,J3%+I3%,8,5 To D%,J2%,J3%+I3%
    Next J%
  Next I%
  Cls
  Print "Maus bewegen und Buttons drücken"
  Do
    Mouse X%,Y%,K%
    Vsync
    Vsync
    Rc_copy S%,X%,Y%,50,50 To D%,X%,Y%,3*Mousek
    ' @Rc_copy(S%,X%,Y%,50,50,D%,X%,Y%,3*Mousek) ! V2.xx-Aufruf
    Vsync
    Vsync
    Rc_copy S%,X%,Y%,50,50 To D%,X%,Y%,3*Mousek
    ' @Rc_copy(S%,X%,Y%,50,50,D%,X%,Y%,3*Mousek) ! V2.xx-Aufruf
  Loop
Endif
```

Version 3.0

RC_INTERSECT() — Überlappung zweier Rechtecke

Var=RC_INTERSECT(Xp1,Yp1,Br1,Hö1,Xp2,Yp2,Br2,Hö2)

Stellt fest, ob sich die beiden angegebenen Rechtecke überschneiden (z.B. für Window-Redraw wichtig). Die Koordinaten und Maße des zweiten Rechtecks **müssen** in 2-Byte- oder 4-Byte-Integervariablen übergeben werden, da diese als Rückgabevariable benötigt werden.

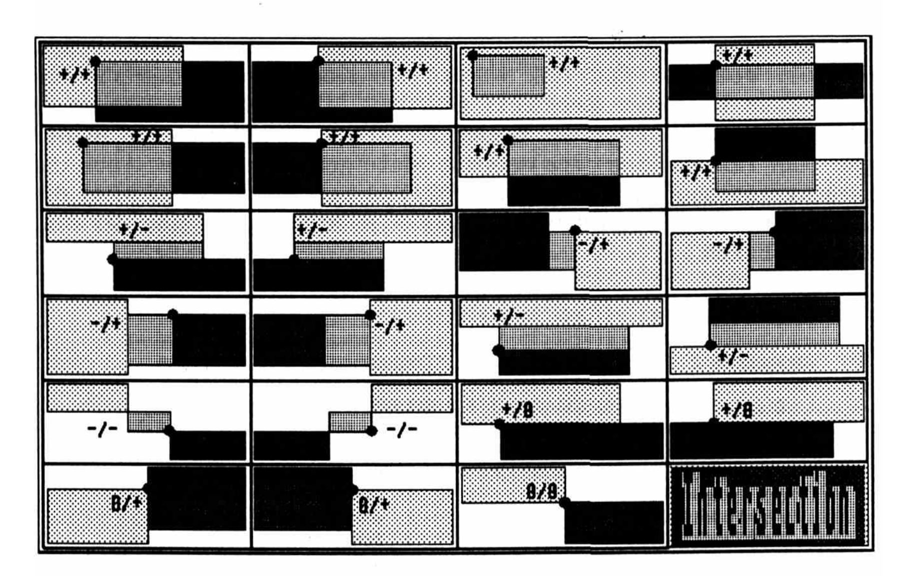
Bei Überschneidung der Rechtecke enthält Var TRUE und in den Variablen Xp2, Yp2, Br2 und Hö2 werden die Koordinaten der linken oberen Ecke, sowie die Breite und Höhe der Schnittfläche zurückgegeben.

Überschneiden sich die Rechtecke nicht, enthält Var den Wert 0 und in Xp2, Yp2, Br2 und Hö2 werden die Koordinate und Maße des dazwischen liegenden Rechtecks geliefert. In diesen Fällen hängt es allerdings von der Lage der beiden Flächen zueinander ab, ob in Br2 und/oder Hö2 positive oder negative Maße geliefert werden. Auch die zurückgegebenen Koordinaten in Xp2/Yp2 beschreiben dann nicht die linke obere Ecke der Fläche, sondern - je nach Fall - eine der drei anderen Ecken.

Beispiel:

```
Graphmode 2                           ! Transparentmodus
Do
  Cls                                 ! Bildschirm löschen
  X1&=Rand(300)+20                    !---.
  Y1&=Rand(60)+20                         !  Zufällige
  Br1&=Rand(300)+20                       !  Koordinaten
  Ho1&=Rand(120)+20                       !- und Maße
  X2&=Rand(300)+100                       !  der beiden
  Y2&=Rand(60)+60                         !  Flächen
  Br2&=Rand(200)+20                       !  holen
  Ho2&=Rand(120)+20                   !---'
  Deffill ,3,1                        ! DEFFILL diagonal1
  Pbox X1&,Y1&,X1&+Br1&,Y1&+Ho1&      ! Box 1 zeichnen
  Deffill ,3,2                        ! DEFFILL diagonal2
  Pbox X2&,Y2&,X2&+Br2&,Y2&+Ho2&      ! Box 2 zeichnen
  Var%=Rc_intersect(X1&,Y1&,Br1&,Ho1&,X2&,Y2&,Br2&,Ho2&)
  Pcircle X2&,Y2&,5                   ! Eckpunkt zeichnen
  Deffill ,2,2                        ! DEFFILL hellgrau
  Pbox X2&,Y2&,X2&+Br2&,Y2&+Ho2&      ! Schnittfläche zeichnen
  ~Inp(2)                             ! Auf Taste warten
Loop
```

Da es nicht immer einfach ist, sämtliche Spielfälle von Flächenüberschneidungen im Kopf zu analysieren, habe ich diese Spielfälle für Sie auf dem Papier durchgespielt.



Version 3.0

SGET { SG } — Bildschirm puffern

SGET Var$

Überträgt den aktuellen Bildschirmspeicher (32000 Byte) nach Var$. Als Bildschirmspeicher wird hier grundsätzlich der "logische" Bildschirm angenommen (siehe XBIOS(3)/(XBIOS(5)). Die String-Variable Var$ braucht nicht vorbereitet zu werden.

Im Gegensatz zu GET 0,0,639,399,Var$ muß hier nicht mit geschnittenen Words operiert zu werden. D.h. zum einen, daß dieser Befehl ca. 15 mal schneller ist als GET, aber auch, daß der Variableninhalt ein anderes Format besitzt (kein Header!). Ein mit SGET gelesener Bildschirm kann deshalb auch nicht durch PUT dirigiert werden.

SGET ist grundsätzlich identisch mit:

```
Var$=Space$(32000)
Bmove Xbios(3),Varptr(Var$),32000
```

Eignet sich in V3.0 auch zur Hintergrund-Screenverwaltung mit RC_COPY.

SHOWM { SH } Mauszeiger anschalten

SHOWM

Der ggfs. durch HIDEM unsichtbare Mauszeiger wird wieder eingeschaltet (auch durch VDISYS 122 möglich). Zum Programmende wird intern immer SHOWM ausgeführt. Auch bei Aufruf von GEM-Formularen (FILESELECT, ALERT etc.) wird automatisch SHOWM ausgeführt, jedoch - im Falle, daß vorher HIDEM gültig war - nach erfolgter Box-Bedienung wieder zurückgenommen.

Beispiel:

```
Print "Maus am unteren Füllrand bewegen"
For I%=0 To 31998 Step 2      !----.
  ' Hidem                           !
  Dpoke Xbios(2)+I%,&HEEEE          ! Bildschirm füllen
  ' Showm                           !
Next I%                       !----'
```

Das Beispiel zeigt, wozu HIDEM intern hauptsächlich verwendet wird. Bei allen Grafikausgaben, bei BMOVE, BITBLT, BLOAD etc. wird für die Dauer des Befehls intern ebenfalls immer HIDEM ausgeführt, da sonst bei Mausbewegungen derselbe Zerstörungseffekt wie im Beipiel eintreten würde. Das ist auch der Grund für das "Flackern" der Maus bei schnell aufeinanderfolgende Grafik-Ausgaben (z.B. bei PUT)

Wenn Sie im obigen Beispiel die beiden REM-Zeilen Hidem und Showm aktivieren, wird der Hintergrund bei Mausbewegungen nicht mehr zerstört, jedoch tritt dann das besagte Mausflackern auf.

SPRITE { SPR } Sprite setzen und löschen

SPRITE Def.var$ [,Xpos,Ypos]

Mit Def.var$ wird eine String-Variable angegeben, deren Inhalt im MKI$-Format die Mausform definiert.

Grundsätzlich sind Sprites nichts anderes als der Mauszeiger, der ja ebenfalls den Charakter eines Sprites besitzt. Der Maus-Sprite wird jedoch automatisch durch die Bewegungen der Maus gesteuert. Mit dem Befehl SPRITE können Sie einen - oder auch mehrere - Sprites definieren und nach Ihren Wünschen auf dem Bildschirm placieren und bewegen.

Der Aufbau eines solchen Sprites ist dem eines Mauszeigers (siehe DEFMOUSE) sehr ähnlich. Die Sprite-Form besteht ebenfalls aus einem Raster mit 16*16 Pixel und es wird ebenfalls ein Format-String gebildet, indem man die Sprite-Daten als MKI$-Werte-Strings aneinanderfügt.

Der Sprite-String hat folgendes Format:

Word 1
X-Koordinate des Aktionspunktes (0 - 15).

Word 2
Y-Koordinate des Aktionspunktes (0 - 15). Bei Sprite-Aktionen beziehen sich Xpos/Ypos auf diesen Aktionspunkt. 0/0 liegt dabei in der oberen linken Ecke und 15/15 in der rechten unteren Ecke des Sprites.

Word 3
Modus (normal = MKI$(0/XOR = MKI$(1)).

Word 4
Maskenfarbe (Hintergrund des Sprites):

```
weiß = MKI$(0)/schwarz = MKI$(1)
```

Word 5
Cursor-Farbe (Sprite-Bild):

```
weiß = MKI$(0)/schwarz = MKI$(1)
```

Word 6 bis 37
Ab dem 6. Word folgen beginnend mit dem Bit-Muster der Sprite-Maske abwechselnd die Zeilen der Maske und die Zeilen der Form (m1-f1-m2-f2-m3-f3-m4-f4 etc.).

Unter Angabe von Def.var$ und der Position Xpos/Ypos wird der Sprite auf dem Bildschirm placiert. Bei jedem Aufruf desselben Sprites mit veränderten Koordinaten wird er an der vorherigen Position

automatisch gelöscht. Soll der Sprite vollständig vom Bildschirm entfernt werden, ist die Angabe der Koordinate zu unterlassen.

Beispiel (Hires/Midres/Lowres):

```
Xt%=2-Sgn(Xbios(4))                  ! X-Auflösung holen
Yt%=Min(2,3-Xbios(4))                ! Y-Auflösung holen
Deffill 1,2,8                        ! DEFFILL hellgrau
Pbox 0,0,290,106                     ! Spritebild...
Color 7                              ! ...
Circle 107,107,7                     ! ...zeichnen
@Dmouse(100,100,8,8,*Sp1$,*Sp2$)! Sprite definieren
Hidem                                ! Maus aus
For I%=0 To 2000 Step 4              !----.
  X%=108+Sin(I%*Pi/180)*(100/Xt%)    !
  Y%=108+Cos(I%*Pi/180)*(40/Yt%)     !- Sprite
  Vsync                              ! darstellen
  Sprite Sp1$,X%,Y%                  !---'
Next I%
Sprite Sp1$                          ! Sprite ausschalten
'
Procedure Dmouse(Mx%,My%,Mxa%,Mya%,Sp1%,Sp2%)
  ' Diese Prozedur ist leer, da sie mit der unter DEFMOUSE
  ' beschriebenen Prozedur Dmouse fast identisch ist. Der
  ' Unterschied besteht in der Zeilen-Zählschleife. Unter DEFMOUSE
  ' sind in Dmouse zwei Schleifenblöcke gesondert hervorgehoben.
  ' Der mit SPRITE gekennzeichnete Block muß für Sprites aktiviert
  ' werden, während der erste - mit MAUS gekennzeichnete - Block
  ' deaktiviert werden muß. Haben Sie das getan, liefert Ihnen die
  ' Prozedur Dmouse auch Sprites nach Maß.
Return
```

SPUT { SPU } — Bildschirm setzen

SPUT Var$

Überträgt die ab VARPTR(Var$) folgenden 32000 Byte in den Bildschirmspeicher. Als Bildschirmspeicher wird hier grundsätzlich der "logische" Bildschirm angenommen (siehe XBIOS(3)/(XBIOS(5)).

SPUT ist grundsätzlich identisch mit:

```
Bmove Varptr(Var$),Xbios(3),32000
```

VSYNC { VS } — VBL-Synchronisation

VSYNC

Wartet mit der nächsten Grafik-Ausgabe auf den nächsten Strahlrücklauf (Vertikal-Blank).

Der Elekronenstrahl beginnt den Bildaufbau in der oberen linken Bildschirmecke und zeichnet dann nacheinander alle Zeilen, bis er in der rechten unteren Ecke angelangt ist. Danach beginnt er den nächsten Aufbau wieder in der linken oberen Ecke. Diesen Weg legt er ca. 70 mal in der Sekunde zurück

Bei Grafikausgaben mit SPUT, SPRITE oder PUT (bzw. in V3.0 mit RC_COPY) kann es sinnvoll sein, den nächsten Bildneuaufbau abzuwarten. Durch die vertikale Synchronisation einer Grafikausgabe mit dem Bildaufbau kann so das Interferenz-Flimmern eingeschränkt werden. Bei Graphikausgaben, die mehr Zeit in Anspruch nehmen, als der Computer zu einem Bildaufbau benötigt, treten allerdings auch dann wieder Interferenzen auf. Ein Beispiel dazu finden Sie unter SPRITE. Die Auswirkung von VSYNC können Sie beobachten, wenn Sie in diesem Beispiel die VSYNC-Zeile löschen.

10. Datenumwandlung

ASC() Textzeichen => ASCII-Wert

Var=ASC("Zeichen")

Ermittelt den ASCII-Wert des Textzeichens Zeichen.

Bei Strings wird nur der ASCII-Wert des ersten Zeichens zurückgegeben. Ist der angegebene String leer (""), wird der Wert 0 geliefert. ASC() bildet die Umkehrfunktion zu CHR$().

Eine Tabelle der möglichen Zeichen und ihrer ASCII-Werte finden Sie unter 24.11 "ASCII-Tabelle". ASCII: American Standard Code for Information Interchange (Deutsch: Amerikanischer Standard-Code für Informationsaustausch).

Beispiel:

```
@Sysfont(1)           ! 8*8-Font einschalten
Print                 ! Leerzeile
For I%=0 To 255       ! Alle 256 ASCII-Zeichen
  Out 5,32            ! Leerzeichen ausgeben
  Out 5,I%            ! ASCII-Zeichen ausgeben
  Print " = ";Asc(Chr$(I%));Spc(5-Len(Str$(I%)));
  ' Diese PRINT-Zeile demonstriert die Konvertierung
  ' eines Wertes in ein Zeichen und anschließend dieses
  ' Zeichen wieder in den ASCII-Wert (ASC(CHR$(I%))).
  ' Ebenso gut könnte die folgende PRINT-Zeile verwendet
  ' werden:
  '          PRINT "=";I%;SPC(5-LEN(STR$(I%)));
Next I%
@Sysfont(2)           ! Standard-Font (8*16) einschalten
'
Procedure Sysfont(Font%)
  ' Auswahl eines System-Fonts für TOS-Text (PRINT, OUT etc.)
  ' Font% = Gewünschter Font
  '         0 = 6*6
  '         1 = 8*8
  '         2 = 8*16
  Local Code$,Adr%               ! Lokale Variablen
  Code$=Mkl$(&HA0002009)+Mki$(&H4E75) ! Maschinen-Code
  '         line_a  0000      ; Line-A 0000 aufrufen
  '         move.l  a1,d0     ; Register A1 nach D0 (long)
  Adr%=Varptr(Code$)          ! Code-Adresse holen
  Dpoke Contrl+2,0            ! CONTRL-
  Dpoke Contrl+6,2            ! Feld...
  Dpoke Contrl+10,102         ! ...belegen
  Lpoke Intin,Lpeek(C:Adr%()+Font%*4) ! Font-Adresse übergeben
```

```
  Vdisys 5
Return
```

Bei der hier übergebenen Adresse handelt es sich um die Startadresse des Font-Headers (!). Es ist dadurch also jederzeit möglich, jeden beliebigen Font als System-Font zu installieren, vorausgesetzt, er hat einen brauchbaren Font-Header, dessen Adresse Sie dann hiermit initialisieren (siehe Kapitel 17.1 "Verwendung eigener Fonts").

BIN$() — Numerisch => Binär

Var$=BIN$(Expr)
Var$=BIN$(Expr [,Stellen]) **(nur V3.0)**

Wandelt Expr in einen Binär-String um. Expr steht für eine(n) beliebige(n) numerische(n) Variable, Konstante, Ausdruck oder Funktion.

In V3.0 wird das Standard-Prefix für Binärzahlen (z.B. %1001110110) erkannt und - zulässige - Wertangaben dieser Art vom Interpreter selbstständig in das GFA-Format (z.B. &X1001110110) umgewandelt.

Außerdem kann in Version V3.0 durch den optionalen Parameter Stellen eine Stellenanzahl (1 - 32) vorgegeben werden, auf die der gewandelte Wert begrenzt wird.

Beispiele:

```
Print Bin$(1273530)
'
Print
For I%=1 To 4            ! 4 Binärwerte
  Read A%                ! Lesen
  Print Bin$(A%),"="'A%  ! Wandeln und ausgeben
Next I%
Data &X1001110,&X100110101,&X10001110,&X100111001
'
Print
For I%=1 To 12           ! 12 Binärwerte
  Read A$                ! Als String lesen
  B$=Right$(String$(14,"0")+A$,14) ! Binär-String auf
  '                      ! 14 Zeichen formatieren
  Print B$,              ! Binär-String ausgeben
  Print "="'Val("&X"+A$) ! Wert ausgeben
  For J%=1 To Len(B$)    ! Alle Zeichen des Strings
    If Val("&X"+B$) And 2^J% ! Bit gesetzt?
      Plot 200+J%,30+I%  ! Dann Punkt setzen
    Endif
  Next J%
Next I%
Data 000111000111000111000
```

```
Data 000111000111000111000
Data 000111000111000111000
Data 111000111000111000111
Data 111000111000111000111
Data 111000111000111000111
Data 000111000111000111000
Data 000111000111000111000
Data 000111000111000111000
Data 111000111000111000111
Data 111000111000111000111
Data 111000111000111000111
```

10.1 Die Zahlensysteme

Es gibt im Computerbereich vier Arten von Zahlensystemen:

- Das dezimale System.
- Das hexadezimale System.
- Das binäre System.
- Das oktale System.

Das allgemein übliche Dezimalsystem hat die Zahl 10 zur Basis (dezi: von lat. decem => zehn/z.B. Dezimeter = 10 Zentimeter). Diese Basis wird jeweils zur Ermittlung der Wertigkeit einer Zahl herangezogen. So werden die Einerstellen aus der Null-Potenz der Zahl 10 ermittelt (10 hoch 0 = 1), die Zehnerstellen aus der Einer-Potenz (10 hoch 1 = 10), die Hunderterstellen aus der Zweier-Potenz (10 hoch 2 = 100) usw.

In den anderen Zahlensystemen ist dieser Potenzierungsvorgang exakt derselbe, nur werden hier andere Zahlen als Basis verwendet. So wird im Hexadezimalsystem (Hexa + Dezi = Hexadezi/6 + 10 = 16) die Zahl 16 als Basis verwendet, im Binärsystem (Bi = 2) die Zahl 2 und im Oktalsystem (Okta = 8) die Zahl 8. So erklären sich die geringen Wertigkeiten der Binärzahlen in ihren Stellen und die hohen Wertigkeiten der Hexadezimalzahlen.

Stellenwertigkeiten:

```
Dezimal:

-------------.--------.---------.---------.--------.
<-...    5.  |   4.   |   3.    |   2.    |   1.   | Stelle
-------------+--------+---------+---------+--------:
<-...  10000 |  1000  |   100   |   10    |   1    | Wert/Format
-------------+--------+---------+---------+--------:
<-...   10^4 |  10^3  |  10^2   |  10^1   |  10^0  | Potenz
-------------'--------'---------'---------'--------'
```

Hexadezimal:

<-...	5.	4.	3.	2.	1.	Stelle
<-...	65536	4096	256	16	1	Wert
<-...	$10000	$1000	$100	$10	$1	Format
<-...	16^4	16^3	16^2	16^1	16^0	Potenz

Binär:

<-...	5.	4.	3.	2.	1.	Stelle
<-...	16	8	4	2	1	Wert
<-...	%10000	%1000	%100	%10	%1	Format
<-...	2^4	2^3	2^2	2^1	2^0	Potenz

Oktal:

<-...	5.	4.	3.	2.	1.	Stelle
<-...	4096	512	64	8	1	Wert
<-...	&10000	&1000	&100	&10	&1	Format
<-...	8^4	8^3	8^2	8^1	8^0	Potenz

Bei den Hexadezimalzahlen gibt es eine Besonderheit zu beachten. Da sich die Zahl 16 nicht mit arabischen Zahlen einstellig darstellen läßt, hat man zu einem Trick gegriffen. Die Hex-Zahlen 0 - 9 werden genauso dargestellt wie im Dezimalsystem. Die Zahlen von 10 bis 15 werden dagegen durch einen Buchstaben repräsentiert (A=10;B=11;C=12; D=13;E=14;F=15). Daher also die Buchstaben in Hexadezimalzahlen.

Die Dezimalzahl 1037 setzt sich also wie folgt zusammen:

```
  (10 hoch 0) * 7 =    7
+ (10 hoch 1) * 3 =   30
  (die dritte Stelle ist nicht besetzt, also: 0)
+ (10 hoch 3) * 1 = 1000
                  ----------
       Summe :        1037
                  ==========
```

Die Binärzahl %100011 ergibt dagegen nach der Wertigkeit ihrer Stellen (immer von rechts gesehen, wie im Dezimalsystem auch):

```
  (2 hoch 0) * 1  = 1
+ (2 hoch 1) * 1  = 2
  (die dritte Stelle ist nicht besetzt, also: 0)
  (die vierte Stelle ist nicht besetzt, also: 0)
  (die fünfte Stelle ist nicht besetzt, also: 0)
+ (2 hoch 5) * 1  = 32
                 ----------
     Summe :        35
                 ==========
```

Das gleiche für die Hexadezimalzahl $F3B0:

```
  (die erste Stelle ist nicht besetzt, also: 0)
  (16 hoch 1) * 11 =   176
+ (16 hoch 2) * 3  =  2816
+ (16 hoch 3) * 15 = 61440
                  ----------
     Summe :         64432
                  ==========
```

Und noch für die Oktalzahl &10537:

```
  (8 hoch 0) * 7   =    7
+ (8 hoch 1) * 3   =   24
+ (8 hoch 2) * 5   =  320
  (die vierte Stelle ist nicht besetzt, also: 0)
+ (8 hoch 4) * 1   = 4096
                  ----------
     Summe :         4447
                  ==========
```

Die Umkehrung dieser Transformation sieht dann so aus: Die Dezimalzahl 1352 soll in das binäre Format verwandelt werden:

```
1352 : 2   =   676    Rest 0    >-----------------------.
 676 : 2   =   338    Rest 0    >---------------------. |
 338 : 2   =   169    Rest 0    >-------------------. | |
 169 : 2   =    84    Rest 1    >-----------------. | | |
  84 : 2   =    42    Rest 0    >---------------. | | | |
  42 : 2   =    21    Rest 0    >-------------. | | | | |
  21 : 2   =    10    Rest 1    >-----------. | | | | | |
  10 : 2   =     5    Rest 0    >---------. | | | | | | |
   5 : 2   =     2    Rest 1    >-------. | | | | | | | |
   2 : 2   =     1    Rest 0    >-----. | | | | | | | | |
   1 : 2   =     0    Rest 1    >---. | | | | | | | | | |
   0 : 2   =     0    Rest 0    >-. | | | | | | | | | | |
                                  | | | | | | | | | | | |
           Die Binärzahl lautet:  0 1 0 1 0 1 0 0 1 0 0 0
```

Nach dem gleichen Schema lassen sich auch Dezimalzahlen in die übrigen beiden Zahlensysteme konvertieren.

Dazu noch das Beispiel für Hexadezimalzahlen: Die Dezimalzahl 35117 soll in das Hexadezimalformat gewandelt werden:

```
35117 : 16 = 2194 => 2194 * 16 = 35104 => Diff. = 13 (D) >---------.
 2194 : 16 =  137 =>  137 * 16 =  2192 => Diff. =  2 (2) >-------. |
  137 : 16 =    8 =>    8 * 16 =   128 => Diff. =  9 (9) >-----. | |
    8 : 16 =    0 =>    0 * 16 =     0 => Diff. =  8 (8) >---. | | |
    0 : 16 =    0 =>    0 * 16 =     0 => Diff. =  0         | | | |
                                                             | | | |
                Die Hexadezimalzahl lautet :                 8 9 2 D
```

Bei derart komfortablen Sprachen wie GFA-BASIC hat man es natürlich nicht mehr nötig, diese Zahlen und Formate "zu Fuß" zu ermitteln. Trotzdem kann es unter Umständen von Vorteil sein, sich damit einigermaßen auszukennen.

Das Hexadezimale Zahlensystem wurde aus einem bestimmten Grund entwickelt. Es lassen sich nämlich "zufälligerweise" genau die Inhalte von vier Bits (Tetrade oder auch Nibble) mit einer Hex-Zahl darstellen. Die Binärzahl 1111 kann maximal den Wert 15 (2^0+2^1+2^2+2^3) annehmen. Und genau dieser Wert läßt sich auch maximal mit einer Hex-Zahl darstellen (F). Ein Byte (8 Bit) wird demnach immer zu einer 2er-Tetrade (2*4 = 8 Bit), ein Word (16 Bit) zu einer 4er-Tetrade (4*4 = 16 Bit) und ein Longword (32 Bit) immer zu einer 8er-Tetrade (8*4 = 32 Bit) zusammengefaßt.

Version 3.0

CFLOAT() — Integerwert => Fließkommawert

Var=CFLOAT(Wert)

Wandelt Wert (Integerwert) in eine Realzahl um. CFLOAT bildet die Umkehrfunktion zu CINT. Diese Funktion ist nur beim V3.0-Compiler von Bedeutung.

CHR$() — ASCII => Textzeichen

Var$=CHR$(Wert)

Liefert ein - dem angegebenen Wert entsprechendes - ASCII-Zeichen. Ist Wert größer als 255, so wird das Zeichen ermittelt, das Wert MOD 256 (bzw. Wert AND 255) entspricht. CHR$ bildet die Umkehrfunktion zu ASC.

Beispiele hierzu finden Sie z.B. unter INKEY$(), LINE INPUT, PRINT, WRITE und ASC()).

Version 3.0

CINT() — Fließkommawert => Integerwert

Var=CINT(Wert)

Wandelt Wert (Realwert) in eine Integerzahl um. Im Gegensatz zu INT wird die Zahl vorher exakt gerundet. CINT bildet die Umkehrfunktion zu CFLOAT.

CVI(), CVL(), CVS(), CVF(), CVD() String => Format-Zahl

Funktion:	Ergebnis:
Var=CVI("2 Zeichen")	16-Bit-Integerwert
Var=CVL("4 Zeichen")	32-Bit-Integerwert
Var=CVS("4 Zeichen")	Realwert (Atari-BASIC-Format)
Var=CVF("6 Zeichen")	Realwert (V2.xx-GFA-BASIC-Format)
Var=CVD("8 Zeichen")	Realwert (MBASIC- oder V3.0-GFA-BASIC-Format)

Es wird die der jeweiligen Funktion entsprechende Zeichenanzahl von x Zeichen in eine Zahl des jeweiligen Formats umgewandelt. Diese Funktionen bilden die Umkehrfunktionen zu MKI$/MKL$/MKS$/MKF$/MKD$.

Beispiel:

```
A%=1932                    ! Beliebiger Wert
A$=Space$(4)               ! String-Puffer
Bmove Varptr(A%),Varptr(A$),4 ! 4 Bytes in Puffer
Print Cvl(A$)              ! Puffer-Inhalt wandeln
```

Weitere Beispiele finden Sie unter INSTR() und in Prozedur Cut unter RIGHT$().

HEX$() — Numerisch => Hexadezimal

Var$=HEX$(Expr)
Var$=HEX$(Expr [,Stellen]) **(nur V3.0)**

Wandelt Expr in einen Hexadezimal-String um. Expr steht für eine(n) beliebige(n) numerische(n) Variable, Konstante, Ausdruck oder Funktion. Durch den optionalen Parameter Stellen kann in V3.0 eine Stellenanzahl (1 - 8) vorgegeben werden, auf die der gewandelte Wert begrenzt wird.

In V3.0 wird außerdem das Standard-Prefix für Hexadezimalzahlen (z.B. $FA5C16) erkannt und - zulässige - Wertangaben dieser Art vom Interpreter selbstständig in das GFA-Format (z.B. &HFA5C16) umgewandelt. Wird als Prefix nur das Und-Zeichen & (z.B. &1EA6F9) verwendet, wird der Ausdruck ebenfalls in den entsprechenden Hex-Wert umgewandelt.

MKI$(), MKL$(), MKS$(), MKF$(), MKD$() — Format-Zahl => String

Funktion:	**Ergebnis:**
Var$=MKI$(16-Bit-Integer-Wert)	**2-Zeichen-String**
Var$=MKL$(32-Bit-Integer-Wert)	**4-Zeichen-String**
Var$=MKS$(Atari-BASIC-Realwert)	**4-Zeichen-String**
Var$=MKF$(V2.xx-GFA-BASIC-Realwert)	**6-Zeichen-String**
Var$=MKD$(MBASIC- oder V3.0-GFA-BASIC-Realwert)	**8-Zeichen-String**

Es wird der in Klammern angegebene Wert in einen, der Wertgröße und dem gewünschten Format entsprechenden String-Ausdruck umgewandelt.

Der Variablenaufbau der verschiedenen Systeme, Interpreter und Compiler kann sich voneinader stark unterscheiden. Um nicht zeitaufwendige Rechenoperationen ausführen zu müssen, die einzelnen Werte anderer Sprachen in das benötigte Format zu übertragen, können diese Funktionen auch dazu verwendet werden, den Datenaustausch zu vereinfachen.

Diese Funktionen bilden die Umkehrfunktionen zu CVI$()/CVL$()/CVS$()/CVF$()/CVD$().

Beispiele hierzu finden Sie unter anderem in der Prozedur Showdat unter LINE INPUT, unter CHAIN und DEFMOUSE, sowie in der Prozedur Cut unter RIGHT$().

OCT$() — Numerisch => Oktal

Var$=OCT$(Expr)
Var$=OCT$(Expr [,Stellen]) **(nur V3.0)**

Wandelt Expr in einen Oktal-String um. Expr steht für eine(n) beliebige(n) numerische(n) Variable, Konstante, Ausdruck oder Funktion.

Will man Integerwerte im Oktal-Format angeben, so kann der Vorsatz &O (z.B.: A%=&O16501) verwendet werden.

Durch den optionalen Parameter Stellen kann in V3.0 eine Stellenanzahl (1 - 11) vorgegeben werden, auf die der gewandelte Wert begrenzt wird.

Weitere Informationen finden Sie in 10.1 "Die Zahlensysteme".

STR$() **Numerisch => String**

Var$=STR$(Wert)
Var$=STR$(Wert [,Stellen [,Real]]) **(nur V3.0)**

Es wird ein Text-String mit der Länge gebildet, die der Anzahl der Ziffern des übergebenen Wertes im Dezimalformat entspricht. Wert kann in jedem beliebigen Zahlensystem angegeben werden. Als Hexadezimal-, Binär- oder Oktalzahl angegebene Werte werden vorher in das Dezimalformat umgewandelt.

Die hiermit erzeugte Ziffernfolge ist keine Zahl mehr, die einen Wert darstellt, sondern lediglich ein String, der die einzelnen Ziffern des Wertes als Textzeichen enthält.

STR$ bildet die Umkehrfunktion zu VAL. Beispiele hierzu finden Sie unter anderem unter BLOAD, BSAVE, in der Prozedur Getdir unter FSNEXT() und in der Prozedur Pcode unter DATA.

Durch den optionalen Parameter Stellen kann in V3.0 eine Stellenanzahl vorgegeben werden (Vor-, Nachkommastellen und ggfs. Dezimalpunkt), auf die der gewandelte Wert begrenzt wird. Der optionale Parameter Real gibt an, auf wieviele Nachkommastellen der gewandelte Wert ggfs. gerundet werden soll. Diese gerundeten Nachkommastellen gehen auf jeden Fall in den gelieferten Wert-String ein, auch wenn Wert eigentlich keine Nachkommastellen beinhaltet.

V3.0-Beispiele:

```
PRINT STR$(572.6169,5,3)   ergibt:   2.617
PRINT STR$(6169,9,5)       ergibt: 169.00000
```

VAL() String => Numerisch

Var=VAL(Var$)

Wandelt alle am Anfang eines Strings stehenden Zeichen, die sich zur Darstellung numerischer Werte eignen, in eine **dezimale** Realzahl um.

Var$ ist eine beliebige Zeichenkette, ein String-Ausdruck oder eine String-Variable, deren Inhalt vom Anfang ausgehend daraufhin untersucht wird, ob Textzeichen enthalten sind, die einen Wert in einem der vier Zahlensysteme darstellen. Die Suche wird abgebrochen, wenn das String-Ende erreicht ist oder die Funktion auf ein Textzeichen trifft, welches nicht wandelbar ist.

Ist das erste Zeichen des Strings ein nicht wandelbares Textzeichen oder der String leer, wird eine Null zurückgegeben.

Beispiele:

```
A$="> ";Str$(123456);" <"
Print A$                   ergibt: > 123456 <
'
A$="1011010 <-Binär"
Print Val("&X"+A$)         ergibt: 90        (= &X1011010)
'
A$="1141331 <-Octal"
Print Val("&O"+A$)         ergibt: 312025    (= &O1141331)
'
A$="AF451DE <-Hexadezimal"
Print Val("&H"+A$)         ergibt: 183783902 (= &HAF451DE)
Print Val("&HEEZeichen")   ergibt: 238           (= &HEE)
'
A$="&X110123"
A%=Val(A$)
Print A%                   ergibt: 13          (= &X1101)
'
Print Val("2.37E+07")      ergibt: 23700000   (= 2.37E+07)
```

VAL?() Anzahl wandelbarer Textzeichen ermitteln

Var=VAL?(Var$)

Ermittelt ab Anfang von Var$ die Anzahl seiner Zeichen, die in numerische Werte konvertiert werden können (siehe VAL()). Var$ steht für eine beliebige Zeichenkette oder String-Variable, die auf die Anzahl ihrer wandelbarer Zeichen untersucht werden soll. Trifft die Funktion auf nicht wandelbare Zeichen, wird die Untersuchung abgebrochen.

Beispiele:

```
Print Val?("237E07")     ergibt: 6  (Exponentialformat)
'
A$="&X110123E"
Print Val?(A$)           ergibt: 6  (inkl. Identifikator)
```

11. Feld-, Speicher- und Zeigeroperationen

11.1 Feldoperationen

ARRAYFILL { ARR } — Feld mit Wert belegen

ARRAYFILL Feld(),Var

Feld bezeichnet ein bereits beliebig dimensioniertes, numerisches Feld (Integer, Real, Boole). Alle Elemente dieses Feldes werden mit dem Wert Var belegt. Var muß mit dem Feldtyp übereinstimmen (z.B. Integer zu Integer).

Beispiel: Umständlich: Dim Feld%(20,32,16)

```
For A%=0 To 20              !---.  Füllt alle
  For B%=0 To 32                |  Elemente des
    For C%=0 To 16              |- Arrays Feld%
      Feld%(A%,B%,C%)=237       |  mit dem Wert
    Next C%                     |  237.
  Next B%                       |
Next A%                     !---'
```

GFA-Methode: Dim Feld%(20,32,16)

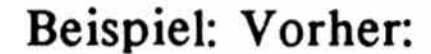
```
Arrayfill Feld%(),237  !----  Tut dasselbe
```

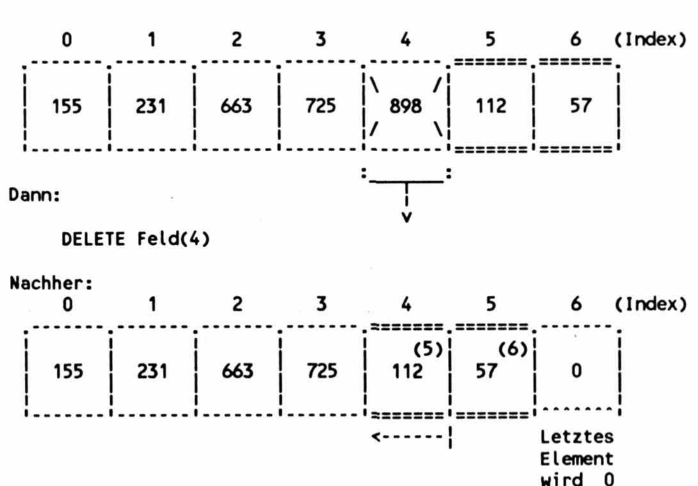
Version 3.0

DELETE { DEL } — Einzelelement aus Feld löschen

DELETE Feld(Index)
DELETE Feld$(Index)

Löscht das einzelne Element Index aus dem Feld Feld, bzw. Feld$. Alle darüberliegenden Elemente werden im Feld um eine Stelle nach unten versetzt. Das letzte Element enthält anschließend den Wert Null, bzw. bei String-Feldern einen Leer-String. DELETE ist die Umkehrung zu INSERT.





DIM { DI } — Feld(er) dimensionieren

DIM Arr1(Ind1 [,Ind2,...]) [,Arr2(Ind1 [,Ind2,...])...]

Legt die Dimension(en) von Arr1 (bzw. Arr2, Arr3 etc.) fest und reserviert hierfür Speicherplatz. Arr steht für beliebige numerische oder alphanumerische Felder.

Ind besagt, wie viele Elemente pro Dimension zugelassen sind. Bei mehrdimensionalen Feldern (z.B. DIM Feld(5,20,7)) ist die Anzahl der Elemente auf 65535, bei eindimensionalen Feldern (z.B. DIM Feld$(100)) nur durch die Größe des Arbeitsspeichers begrenzt.

Beispiele hierzu finden Sie unter anderem unter CHAIN, RECALL#, RCALL, PI und SETCOLOR.

Achtung: Bei großen Dimensionierungen kann sich die Adressenlage der übrigen Variablen - insbesondere bei String-Variablen - verschieben. Im Falle, daß Maschinenprogramme in String-Variablen abgelegt wurden und das Programm ohne vorherige VARPTR-Abfrage, aufgerufen wird führt dies dann ggfs. zum Absturz. Zur Speicherung der Routine eignet sich in den V2.xx-Versionen am besten ein 4-Byte-Integerfeld:

```
DIM A%(Codelänge/4)
Start%=VARPTR(A%(0))
BLOAD "MASCHINE.COD",Start%
CALL Start%
```

In der V3.0-Version ist dieses Problem durch INLINE (siehe dort) aus der Welt geschafft.

11.1.1 Aufbau eines mehrdimensionalen Feldes

Stellen Sie sich bitte einen Schrank vor. Dieser Schrank wird nun in zwei Hälften geteilt. Eine rechte und eine linke Hälfte. Innerhalb der Hälften sind Schubladen untergebracht. Nehmen wir an, jede Hälfte besitzt zwei Schubladen.

Diese Schubladen werden nun wiederum in einzelne Fächer unterteilt. Jede Schublade erhält drei Fächer. Wir haben nun also einen Schrank mit 12 Fächern (2*2*4) eingerichtet.

Die entsprechende Dimensionierung dazu:

```
DIM Schrank(1,1,2)          -> bei OPTION BASE 0
DIM Schrank(2,2,3)          -> bei OPTION BASE 1
```

Sie wundern sich evtl., warum im ersten Fall nicht (2,2,3) steht. Das hat den Grund, daß Arrays **immer** mit dem Index 0 zu zählen beginnen, falls **nicht** durch OPTION BASE 1 das Null-Element eliminiert wurde. So wird im ersten Fall vorausgesetzt, daß als jeweils kleinster Index einer Dimension ein Null-Element vorhanden ist.

Der angegebene Index bedeutet dann: "dimensioniere **bis** Element X",

```
Also: Element 0 und 1 der ersten Dimension      --.
      Element 0 und 1 der zweiten Dimension       |-> DIM(1,1,2)
      Element 0, 1 und 2 der dritten Dimension --'
```

Im folgenden gehe ich grundsätzlich davon aus, daß OPTION BASE 0 aktiv ist. In diesem Schrank sollen nun verschiedene "Dinge" untergebracht werden. In unserem Fall sind dies Zahlen (hier: Mengenwerte). Zur besseren Orientierung ordne ich den beiden Schrankseiten erst einmal Begriffe zu:

```
Linke Schrankseite    ->  Werkzeug  (1D-Index=0)
Rechte Schrankseite   ->  Bücher    (1D-Index=1)

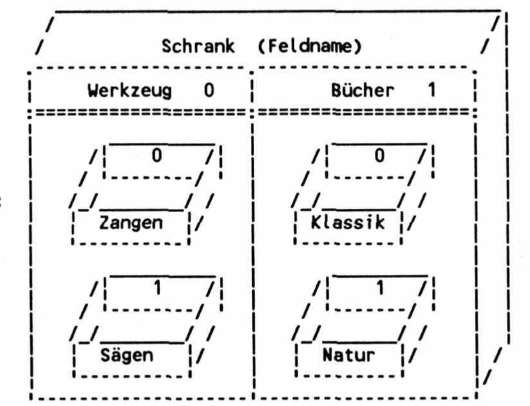
                       ______________________________
                      /            Schrank         /|
                     .--------------.-------------. |
                     |              |             | |
   1. Dimension:     |   Werkzeug   |   Bücher    | |
                     |              |             |/
                     '--------------'-------------'
```

Den Schubladen der Werkzeug-Seite gebe ich die Aufschriften:

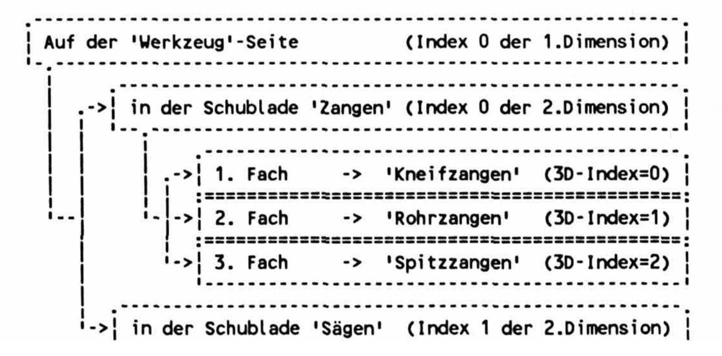
```
1. Schublade  ->  Zangen  (2D-Index=0)
2. Schublade  ->  Sägen   (2D-Index=1)
```

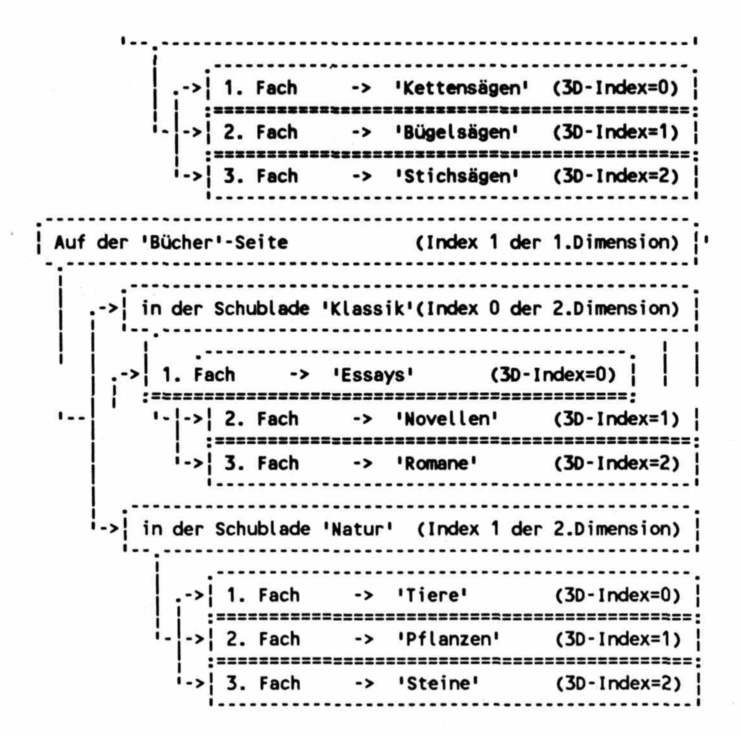
Die Schubladen der Bücher-Seite bekommen die Namen:

```
1. Schublade  ->  Klassik (2D-Index=0)
2. Schublade  ->  Natur   (2D-Index=1)
```



Nun gebe ich noch den Elementen einen Namen:





Bis hierhin sieht das vielleicht etwas übertrieben aus. Der Sinn der Sache ist aber der, daß nun anhand von sogenannten "Indizes" auf jedes einzelne Fach der untersten Ebene zugegriffen werden kann.

Unter Index versteht man allgemein ein Unterscheidungsmerkmal bzw. eine Kennzeichnung gleichartiger Größen, die dann statt durch ihren Namen durch einen ihnen zugewiesen Tabellenwert (Tabellenposition = Index) identifiziert werden können.

```
z.B.  Oberbegriff 'Tier'
      ------------------
          1 = 'Haus-'
          2 = 'Nutz-'
          3 = 'Raub-'
          4 = 'Schalen-'
          5 = 'Stachel-'
```

Statt des Begriffs 'Raubtier' könnte man nun auch 'Tier(3)' sagen und dann - um die Bedeutung von (3) zu erfahren - in der Tabelle unter dem Index 3 nachschauen.

Stellt man sich für die Elemente 'Pflanzen', 'Romane' usw. in der untersten Dimension Speicherplätze im Computer vor, so kann man sich nun durch die Angabe der Indizes jederzeit Informationen über den Inhalt der Plätze verschaffen. Dieses ist vor allem dann wichtig, wenn Informationen abgelegt werden, die bestimmten 'Familien' zugeordnet werden sollen.

Um nun z.B. den Inhalt des Faches "Pflanzen" zu ermitteln, kann man nun durch Ausgaben (z.B. PRINT Schrank(1,1,0)), eine beliebige Zuweisung (z.B. A%=Schrank(1,1,0)) oder Einbindung in einen Ausdruck (z.B. IF 2*Schrank(1,1,0)+10) dieses Fach lesen oder durch Schrank(1,1,0)=XYZ etwas darin ablegen.

Um die Identifikation der einzelnen Schrankseiten, Schubladen und der darin enthaltenen Fächer zu vereinfachen, kann man den numerischen Indizes auch Variablennamen zuweisen:

```
Werkzeug=0
            Zangen=0
                        Kneifzangen=0
                        Rohrzangen=1
                        Spitzzangen=2
            Saegen=1
                        Kettensaegen=0
                        Buegelsaegen=1
                        Stichsaegen=2
Bücher=1
            Klassik=0
                        Essays=0
                        Novellen=1
                        Romane=2
            Natur=1
                        Tiere=0
                        Pflanzen=1
                        Steine=2
```

Jetzt fällt es natürlich leicht, z.B. gezielt in Erfahrung zu bringen, wieviele Bücher zum Thema "Pflanzen" in meiner Bibliothek stehen:

```
IF Schrank(Buecher,Natur,Pflanzen)=0
  PRINT "Natur-Banause!"
ENDIF
```

oder:

```
IF Schrank(Werkzeug,Zangen,Rohrzangen)>10
  PRINT "Na dann: fröhlichen Rohrbruch!"
ENDIF
```

Ich hoffe, daß Ihnen dieser Ausflug in die Dimensionen in Zukunft etwas dabei helfen wird, die - manchmal sehr - verzwickten Zusammenhänge bei mehrdimensionalen Feldern zu verstehen. Bei dem hier durchgespielten Beispiel habe ich die Dimensionenstiefen in Grenzen gehalten, die sich noch realtiv leicht nachvollziehen lassen. Stellen Sie sich zum Schluß bitte noch einmal etwas vor:

Ein Gelände mit 100 Lagerhallen. In allen Lagerhallen stehen 60 Regal-Blöcke zu je 50 Regalen. Jedes Regal ist vertikal in 200 Spalten unterteilt und jede dieser Spalten in 40 Regalböden.

Und zu guter Letzt stehen auf jedem Regalboden noch 20 Kästen, wovon wiederum jeder in 10 Fächer aufgeteilt ist.

```
DIM Lager(100,60,50,200,40,20,10)
```

Die sich daraus ergebende Elementeanzahl ist zwar auf einem ST nicht zu verarbeiten (100*60*50*200*40*20*10 = 480 Milliarden), aber Sie gibt einen Ausblick auf die fast unbegrenzten Einsatzmöglichkeiten von mehrdimensionalen Feldern.

DIM?() — Menge der Feldelemente ermitteln

Var=DIM?(Feld())

Feld ist ein beliebiger numerischer oder alphanumerischer Feldname. DIM? liefert die Anzahl aller Elemente dieses Feldes. Bei nicht dimensionierten Feldern wird der Wert 0 geliefert.

Beispiel:

```
Dim Feld%(2,3,4)
If Dim?(Feld%())>0
   Print "Das Feld hat ";Dim?(Feld%());" Elemente"
Else
   Print "Das Feld ist nicht dimensioniert!"
Endif
```

ERASE { ER } Feld(er) löschen

V3.0: { ERA }

ERASE Feld()
ERASE Feld1() [,Feld2() [,...]] **(nur V3.0)**

Feld bezeichnet ein beliebiges Feld, das gelöscht werden soll. Die Dimensionierung wird aufgehoben und der dafür reservierte Speicherplatz wieder freigegeben.

In der Version V3.0 ist es auch möglich, eine Liste von Feldern anzugeben, die dann mit nur einem Befehl gelöscht werden.

Felder, die nach ihrer Verwendung nicht mehr benötigt werden, sollten sofort wieder mit diesem Befehl gelöscht werden, um den durch sie belegten Speicherplatz wieder dem Programm zur Verfügung zu stellen.

Eine wichtige Einsatzmöglichkeit ist, wenn Felder in allgemein verwendbaren Prozeduren (Utilities) eingesetzt werden sollen. Da nicht erwartet werden kann, daß für jedes Utility die Dimensionierungen im Hauptprogramm vorgenommen werden, sollten diese also in der Prozedur selbst erfolgen. Nach Abschluß der Arbeiten in dieser Prozedur wird das Feld wieder gelöscht, um beim nächsten Aufruf wieder neu dimensioniert werden zu können.

Beispiele finden Sie unter anderem unter RECALL, in Prozedur Cbox unter PI und in der Prozedur Menue unter GET.

Nur Version 3.0

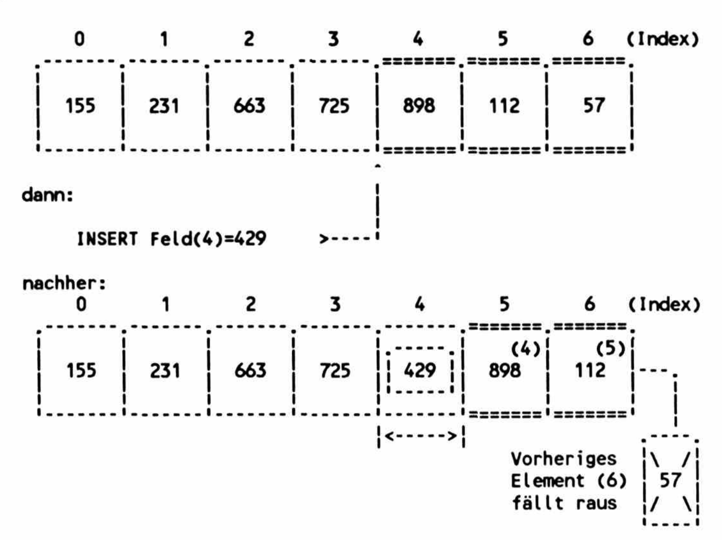
INSERT { INS } Einzelelement in Feld einfügen

INSERT Feld(Index)=Wert
INSERT Feld$(Index)=Text

Fügt das einzelne Element Index in das Feld Feld bzw. Feld$ mit dem zugewiesenen Wert bzw. String ein. Alle darüberliegenden Elemente werden um eine Stelle nach oben versetzt. Das letzte Element des Feldes wird dabei aus dem Feld entfernt.



Beispiel: Vorher:



OPTION BASE { OPT BASE } Feld-Basiselement bestimmen

OPTION BASE Basis

Basis bestimmt das Basis-Element aller Felder (0 oder 1). Die Basis kann im Programm mehrmals geändert werden, da sich die schon definierten Elemente dem neuen Index anpassen (z.B. OPTION BASE 1 -> A$(0) wird A$(1), A$(1) wird A$(2) etc.).

Wurde ein Feld z.B. unter OPTION BASE 0 mit Dim Feld(5) eingerichtet, so ergibt Print Feld(6) eine Fehlermeldung und das Element Feld(0) ist ohne weiteres ansprechbar. Wird jedoch anschließend OPTION BASE 1 verfügt, so ist das vorher letzte Element Feld(5) nun ohne Fehlermeldung mit Feld(6) ansprechbar und das vorher erste Element Feld(0) ergibt eine Fehlermeldung.

Die Basis-Bestimmung wirkt sich auf alle dimensionierten Felder gleichzeitig aus.

Nur Version 3.0

QSORT { QS } — Feld (-Bereich) Quick-Sortierung

QSORT Feld([Sign]) [,Anz [,Feld2%()]]
QSORT Feld$([Sign]) [WITH Vorgabe()] [,Anz [,Feld2%()]]

Es können Felder nach ihrer numerischen Größe oder alphabetischer Reihenfolge sortiert werden. Feld() ist dabei ein numerisches Feld beliebigen Typs. Feld$() ist ein String-Feld.

Das optionale "Sign" (innerhalb der Leerklammer, z.B. QSORT Feld(+)) ist entweder ein Plus- oder Minuszeichen, daß die Sortierrichtug angibt. Sollen die Werte, bzw. Strings mit dem höchsten Wert, bzw. Buchstaben in Element 0 beginnend absteigend sortiert werden, ist ein Minuszeichen einzusetzen. Das Pluszeichen oder keine Sign-Angabe bewirkt die aufsteigende Sortierung.

Anz kann optional verwendet werden, um eine Elementanzahl zu bestimmen, bis zu welcher sortiert werden soll (z.B. 6 = von 0 - 5 bei OPTION BASE 0, bzw. von 1 - 6 bei OPTION BASE 1). Außerdem kann optional ein 4-Byte-Integerfeld (Feld2%()) angegeben werden, dessen Elemente unabhängig von ihrem Inhalt parallel mit dem eigentlichen Sortierfeld mitsortiert werden.

Bei String-Feldern kann durch WITH zusätzlich ein beliebiges Integerfeld (Vorgabe() => 1-, 2- oder 4-Byte-Integer) mit mindestens 256 Elementen bestimmt werden, dessen Elemente-Inhalte die Reihenfolge der Sortierung vorgibt. Wären z.B. alle 256 Vorgabe()-Elemente mit den ASCII-Werten in normaler Reihenfolge belegt (0-255), so kann Vorgabe() vernachlässigt werden. Werden dagegen z.B. die Zeichen a und A vertauscht, so steht A in der Sortierfolge über a (normalerweise umgekehrt), während alle anderen Zeichen normal sortiert werden. So kann eine völlig willkürliche Sortierfolge vorgegeben werden (siehe 24.11 "ASCII-Tabelle").

Beispiel 1:

```
Dim Feld%(20)              ! DIM Feld
Print "unsortiert:"
For I%=0 To 20             ! 20 mal
  Feld%(I%)=Rand(100)      ! Zufällige Werte zuweisen
  Print Feld%(I%)          ! Und ausgeben
Next I%                    ! Nächstes Element
Qsort Feld%()              ! Feld sortieren
Print At(15,1);"steigend sortiert:"
For I%=0 To 20             ! 20 sortierte Elemente
```

```
  Print At(15,I%+2);Feld%(I%) ! Neu ausgeben
Next I%
```

Beispiel 2:

```
Dim Feld%(20)           ! DIM Feld
Print "unsortiert:"
For I%=0 To 20          ! 20 mal
  Feld%(I%)=Rand(100)   ! Zufällige Werte zuweisen
  Print Feld%(I%)       ! Und ausgeben
Next I%                 ! Nächstes Element
Qsort Feld%(-),5        ! 5 Elemente "fallend" sortieren
Print At(15,1);"fallend sortiert:"
For I%=0 To 20          ! 20 Elemente
  Print At(15,I%+2);Feld%(I%) ! Neu ausgeben
  If I%<5               ! Im Sort-Bereich?
    Print At(20,I%+2);"<-- sortiert"
  Else
    Print At(4,I%+2);"--------->"
  Endif
Next I%
```

Beispiel 3:

```
Dim Feld%(20),Feld2%(20) ! DIM Feld und Parallelfeld
Print "unsortiert:      Indexfeld:"
Print Spc(27);"¦"
For I%=0 To 20          ! 20 mal
  Feld%(I%)=Rand(100)   ! Zufällige Werte zuweisen
  Feld2%(I%)=I%         ! Parallelfeld indizieren
  Print Feld%(I%),Feld2%(I%) ! Und ausgeben
Next I%                 ! Nächstes Element
Qsort Feld%(+),15,Feld2%() ! 15 Elemente + Index sortieren
A$="¦ steigend sortiert:    Indexfeld (mitsortiert):"
Print At(28,1);A$
For I%=0 To 20          ! 20 Elemente
  Print At(28,I%+3);"¦ ";Feld%(I%) ! Feld neu ausgeben
  Print At(52,I%+3);Feld2%(I%) ! Index neu ausgeben
  If I%<15              ! Im Sort-Bereich?
    Print At(34,I%+3);" -- sortiert --"
  Else
    Print At(34,I%+3);" - unsortiert -"
  Endif
Next I%
```

Beispiel 4:

In den ersten 3 Beispielen können Sie statt des 4-Byte-Integerfeldes Feld%() ebenso gut andere numerische Feldtypen oder ein String-Feld einsetzen. Der Sortiermodus und das Ergebnis würden dabei prinzipiell gleich bleiben, nur daß bei String-Feldern nicht nach numerischer Reihenfolge sondern nach der Reihenfolge der den Zeichen entsprechenden ASCII-Werten sortiert werden würde. Das folgende Beispiel demonstriert das Sortieren nach einer beliebigen Reihenfolge.

```
Sysfont(1)                    ! 8*8Font setzen (siehe unten)
Dim Feld$(40),Feld2%(40),Vorgabe¦(256) ! DIMs
Print "unsortiert:     Indexfeld:"
Print Spc(27);"¦"
X$="äÄöÖüÜß"                  ! Umlaute und Eszet
X2$="AAOOUUS"                 ! Ersatzkriterium
For I%=0 To 255               ! Alle ASCIIs
  Vorgabe¦(I%)=I%             ! In Sortiervorgabe einsetzen
  If Instr("abcdefghijklmnopqrstuvwxyz",Chr$(I%))
    '                         ! Kleinbuchstaben?
    Vorgabe¦(I%)=Asc(Chr$(I%-32)) ! Durch Großbuchstaben
    '                         ! Ersetzen
  Endif
  If Instr(X$,Chr$(I%))       ! Umlaut oder Eszet?
    Vorgabe¦(I%)=Asc(Mid$(X2$,Instr(X$,Chr$(I%)),1))
    '                         ! Durch Ersatzkriterium
  Endif                       ! Ersetzen
Next I%
For I%=0 To 40                ! 40 mal
  For J%=0 To 8               ! 8 Zeichen je String
    X%=Rand(58)+65            ! Zufalls-ASCII
    '         A-Z = 26 Zeichen
    '         a-z = 26 Zeichen
    '     ÄÖÜäöüß =  7 Zeichen
    '               ------------
    '   insgesamt = 59 Zeichen.
    '
    ' Rand(59) wählt also einen Wert zwischen 0 und 58.
    ' Dieser Wert wird anschließend um 65 erhöht
    ' -> ASCII-Wert von 'A' ist 65.
    If X%>115                 ! X% größer A-Z und a-z?
      X%=Asc(Mid$(X$,X%-116,1))! Dann Umlaut oder Eszet
    Else if X%>90             ! X% zwischen a-z?
      Add X%,6                ! X%+6 = mindesten Asc("a")
    Endif
    Feld$(I%)=Feld$(I%)+Chr$(X%) ! Zeichen einbinden
  Next J%
  Feld2%(I%)=I%               ! Parallelfeld indizieren
  Print Feld$(I%),Feld2%(I%)  ! Und ausgeben
Next I%                       ! Nächstes Element
Qsort Feld$() With Vorgabe¦(),41,Feld2%()
' Alle Elemente + Index nach Vorgabe steigend sortieren
A$="¦ steigend sortiert:  Indexfeld (mitsort.):"
Print At(28,1);A$
For I%=0 To 40                ! 20 Elemente
  Print At(28,I%+3);"¦ ";Feld$(I%) ! Feld neu ausgeben
  Print At(50,I%+3);Feld2%(I%) ! Index neu ausgeben
Next I%
Sysfont(2)
Procedure Sysfont(Font%)
  ' Hier bitte die Prozedur Sysfont einbinden (siehe ASC()).
Return
```

Version 3.0

SSORT {SS} — Feld (-Bereich) Shell-Sortierung

SSORT Feld([Sign]) [,Anz [,Feld2%()]]
SSORT Feld$([Sign]) [WITH Vorgabe()] [,Anz [,Feld2%()]]

Erläuterungen zu QSORT gelten hier analog (siehe dort).

Der wesentliche Unterschied zwischen beiden Sortier-Algorithmen ist, daß QSORT rekursiv arbeitet und dadurch wesentlich mehr Speicherplatz benötigt als das SSORT-Verfahren. Dafür ist SSORT in den meisten Fällen wesentlich langsamer.

Das Shellsort-Verfahren (es wurde von einem Engländer namens Shell entwickelt) ist aufgrund seines Aufbaus eher dazu geeignet, schon vorsortierte Felder zu sortieren, was beim Quicksort-Verfahren unter Umständen sogar eine Zeitverzögerung bewirken kann. In den meisten Fällen hängt es jedoch von Ihrem persönlichen "Geschmack", dem noch verfügbaren Speicher und der jeweiligen Situation ab, welcher Sortierbefehl für Sie in Frage kommt.

Für die V2.xx-Versionen möchte ich hier einen Sortier-Algorithmus vorstellen, der aufgrund seiner nicht gerade atemberaubenden Geschwindigkeit von den meisten Profis gemieden wird. Für den Hausgebrauch ist er dagegen oft sehr nützlich, vor allem weil er spielend zu durchschauen ist und leicht an verschiedene Situationen angepaßt werden kann.

```
Files "\*.*" To "\FILEDAT.LST" ! Disk-Verzeichnis
Dim A$(200)                    ! DIM Aufnahme-Feld
Open "I",#1,"FILEDAT.LST"      ! Datei öffnen
While Eof(#1)=0                ! Noch nicht EOF?
  Inc I%                       ! Zähler +1
  Line input #1,A$(I%)         ! Zeile lesen
Wend
Close #1                       ! Datei schließen
@Sort(*A$(),I%)                ! Feld sortieren
For J%=0 To I%                 ! Alle sortierten Zeilen
  Print A$(J%)                 ! Ausgeben
Next J%
'
Procedure Sort(Ptr%,Lim%)
  ' Ptr% = Pointer auf das zu sortierende String-Feld
  ' Lim% = Feldindex, bis zu welchem sortiert werden soll
  '
  Local J%,K%                  ! Lokale Variablen
  Dim Puffer$(1)               ! DIM Dummy-Swap-Feld
  Swap *Ptr%,Puffer$()         ! Felder tauschen
  For J%=0 To Lim%-1           ! Laufindex 1
```

```
        For K%=J%+1 To Lim%        ! Laufindex 2 (für alle
          'Elemente ab aktuellem Index der 1. Schleife)
          If Upper$(Puffer$(J%))>Upper$(Puffer$(K%))
            'Index1-Element größer Index2-Element?
            Swap Puffer$(J%),Puffer$(K%) ! Dann tauschen
          Endif
        Next K%
      Next J%
      Swap *Ptr%,Puffer$()         ! Felder wieder tauschen
      Erase Puffer$()              ! Dummy-Swap-Feld löschen
    Return
```

Der Algorithmus läßt sich auch für numerische Werte verwenden, wenn vorher die Feldtypen entsprechend geändert werden. Außerdem ist dann der Einsatz der beiden UPPER$()-Funktionen überflüssig.

11.2 Speicheroperationen

Version 3.0

ABSOLUTE { AB } Variable auf Adresse setzen

ABSOLUTE Var,Adresse

Die Adresse der numerischen Variablen Var (beliebiger Typ) wird auf die absolute Speicheradresse Adresse gelegt.

Adresse ist in jedem Fall identisch mit VARPTR(Var):

```
ABSOLUTE Var%,9952
PRINT 9952=V:Var%
     -> Ausgabe = -1 (TRUE)
```

Eine Zuweisung zu einer ABSOLUTE-Variablen ist gleichbedeutend mit:

```
POKE Adresse,Bytewert   ->  Bei Byte-Variablen Var|
DPOKE Adresse,Wordwert  ->  Bei Word-Variablen Var&
LPOKE Adresse,Longwert  ->  Bei Long-Variablen Var%
```

Die Abfrage einer ABSOLUTE-Variablen ist gleichbedeutend mit:

```
Bytewert=PEEK(Adresse)  ->  Bei Byte-Variablen Var|
Wordwert=DPEEK(Adresse) ->  Bei Word-Variablen Var&
Longwert=DPEEK(Adresse) ->  Bei Long-Variablen Var%
```

Bei Real- oder Boole-Variablen ist das jeweilige Zahlenformat zu beachten. ABSOLUTE mit Feld- und String-Variablen ist nicht möglich. ABSOLUTE ist innerhalb von PROCEDUREs und FUNCTIONs mit

zuvor als lokal definierten Variablen ebenso möglich, wie auf globaler Ebene. Nach Rückkehr zum Hauptprogramm wird die "Absolutierung" jedoch wieder aufgehoben.

Statt des Kommas zwischen Var und Adresse kann auch ein Gleich-Zeichen (ABSOLUTE Var%=Adresse) verwendet werden.

Variablen, die auf Systemvariablen-Adressen im Supervisorbereich (<2048) gelegt werden, können anschließend weder gelesen noch belegt werden. Bei Systemvariablen außerhalb des Supervisorbereichs ist Vorsicht walten zu lassen, da bei irrtümlichen oder unzulässigen Zuweisungen Abstürze die frühere oder spätere Folge sein können.

Dieser Befehl eignet sich vorzüglich dazu, Speicherplätze zu analysieren, die unabhängig vom BASIC-Programm durch das TOS oder GEM verändert werden. Hier bieten sich vor allem bestimmte Systemvariablen, die Line-A-Variablen, die verschiedenen GEM-Arrays sowie die VDIBASE an.

Beispiel 1:

```
Print "<Control> und/oder <Shift> und/oder ";
Print "<Alternate> und/oder <CapsLock> drücken"
Absolute Switch¦,3611         ! Umschalttasten-Status (Byte)
Do
  Print at(10,10);Switch'''   ! abfragen
Loop
```

Aus der Byte-Adresse 3611 kann ständig der aktuelle Status der Umschalttasten <Shift>/<Control>/<Alternate>/<CapsLock> ausgelesen werden. Es handelt sich um einen 5-Bit-Vektor:

```
Bit 0  =  <Shift rechts>      Bit 1  =  <Shift links>
Bit 2  =  <Control>           Bit 3  =  <Alternate>
Bit 5  =  <CapsLock>
```

Durch Switch|=Status kann auch ein beliebiger Status gesetzt werden. Dieser Status bleibt solange erhalten, bis eine der als "gedrückt" geltenden Tasten das nächste Mal gedrückt oder ein neuer Status gesetzt wird.

```
Absolute Switch¦,3611         ! Umschalttasten-Status (Byte)
Print "Maus kann durch die Cursor-Tasten bewegt werden"
Print "Abbruch durch <Alternate>"
Switch¦=8
Repeat
Until (Switch¦ And 8)=0
```

Beispiel 2:

```
Absolute Media&,2482  ! Media-Change-Flag (Byte)
Media&=True           ! setzen
```

Ein gesetztes Media-Change-Flag veranlaßt das System dazu, beim nächsten Diskettenzugriff das aktuelle Direktory auch dann zu laden, wenn zwischenzeitlich die Diskette nicht gewechselt wurde (wichtig bei Direktory-Zugriffen durch BIOS(4)).

Beispiel 3:

```
Absolute Hmaus|,3583            ! Horiz. Mausbewegung (Byte)
Absolute Vmaus|,3584            ! Vert. Mausbewegung  (Byte)
Print "Bitte Maus bewegen"
Do
  Repeat
  Until Vmaus| Or Hmaus|       ! Auf Bewegung warten
  Cls                          ! Bildschirm löschen
  If Hmaus|>0 And Hmaus|<16    ! Maus nach rechts?
    Print At(13,10);           ! Cursor positionieren
    Out 5,3                    ! Rechts-Pfeil
  Endif
  If Hmaus|>15 And Hmaus|<256 ! Maus nach links?
    Print At(9,10);            ! Cursor positionieren
    Out 5,4                    ! Links-Pfeil
  Endif
  If Vmaus|>0 And Vmaus|<16    ! Maus nach unten?
    Print At(11,11);           ! Cursor positionieren
    Out 5,2                    ! Abwärts-Pfeil
  Endif
  If Vmaus|>15 And Vmaus|<256 ! Maus nach oben?
    Print At(11,9);            ! Cursor positionieren
    Out 5,1                    ! Aufwärts-Pfeil
  Endif
  Clr Hmaus|,Vmaus|              ! Adressen auf Null setzen
Loop
```

Die oben eingesetzten Adressen 3583 und 3584 geben Auskunft über die zuletzt ausgeführte Mausbewegung. Die Bits geben außerdem in 4 Stufen an, wie schnell die Maus in die jeweilige Richtung bewegt wurde:

```
Bit 0 von Hmaus| gesetzt    => langsam nach rechts
 bis                           ...
Bit 3 von Hmaus| gesetzt    => schnell nach rechts
Bit 4 von Hmaus| gesetzt    => schnell nach links
 bis                           ...
Bit 7 von Hmaus| gesetzt    => langsam nach links
Bit 0 von Vmaus| gesetzt    => langsam nach unten
 bis                           ...
Bit 3 von Vmaus| gesetzt    => schnell nach unten
```

```
Bit 4 von Vmaus¦ gesetzt      => schnell nach oben
 bis                             ...
Bit 7 von Vmaus¦ gesetzt      => langsam nach oben
```

Beispiel 4:

```
Absolute Mmove¦,10206        ! Letzte Mausaktion (Byte)
Print "Bitte Maus bewegen und/oder Knöpfe drücken"
Clr Mmove¦                   ! Adresse löschen
Do
  Repeat
  Until Mmove¦               ! Auf Mausaktion warten
  Cls
  Print "zuletzt:"
  If (Mmove¦ And 128)>0      ! Maustaste rechts?
    Print "Button rechts"
  Else if (Mmove¦ And 64)>0  ! Maustaste links?
    Print "Button links"
  Endif
  If (Mmove¦ And 32)>0       ! Maus bewegt?
    Print "Mausbewegung"
  Endif
  Pause 20
Loop
```

Durch die Byte-Adresse 10206 kann die letzte Mausaktion ermittelt werden. Der Adreßinhalt bleibt bis zur nächsten Aktion erhalten. Die letzte Aktion kann auch nach längerer Zeit noch ausgelesen werden.

Beispiel 5:

```
Absolute Mword%,10226          ! Word-Position des Mauszeigers
Do
  Print At(10,10);Mword%       ! Mauszeiger-Word anzeigen
Loop
```

Adresse 10226 (Longword) liefert die absolute Word-Adresse, auf welcher sich der Aktionspunkt der Maus im Bildschirmspeicher aktuell befindet.

BITBLT { BIT } — Speicherbereiche verknüpfen

In V3.0: { BI }

BITBLT Q_raster%(),Z_raster%(),R_def%() (VDI)
BITBLT Blockadresse% (Line-A -> nur V3.0)
BITBLT Parameterfeld%() (Line-A -> nur V3.0)

Zu Syntax-Variante 1 (VDI-BITBLT)

Dies ist eine Direkt-Zugriffsmöglichkeit auf eine komplizierte VDI-Funktion (COPY RASTER). Die gebräuchlichste Verwendung ist das Verschieben von Bildschirmbereichen auf dem Bildschirm oder in den Speicher (siehe GET/PUT). Dabei kann bestimmt werden, in welcher Art das copierte Raster sich auf den Zielbereich auswirken soll (siehe PUT). Es können allerdings auch beliebige Speicherbereiche auf diese Art in einem Raster definiert und verknüpft werden.

Es sind drei Integer-Felder vorzubereiten. Das erste Q_raster%() beschreibt in 6 Parametern das Quellraster, während das zweite Feld Z_raster%() die gleichen 6 Parameter für das Zielraster enthält. Durch die 9 Parameter-Elemente des dritten Feldes R_def%() können die beiden Rechtecke (Bildschirmbereich oder idealisiertes Speicher-Rechteck) in Punkten, bzw. Bits bestimmt werden. Das Zielraster Z_raster%() wird in einem beliebigen Modus mit dem Quellraster Q_raster%() maskiert, bzw. modifiziert. R_def%() bezeichnet den zu verknüpfenden Bereich der beiden Raster.

Es sind drei Integer-Felder vorzubereiten:

```
Q_raster%()
(0)      Quellraster-Adresse (muß gerade sein).
(1)      Quellraster-Breite in Punkten (durch 16 teilbar!).
(2)      Quellraster-Höhe in Punkten (Zeilenanzahl).
(3)      Quellraster-Breite in Words (immer: Q_raster(1)/16 ).
(4)      Quellraster-Format (immer 0).
(5)      Anzahl der Quellraster-Bit-Planes:
         (Hires=1/Midres=2/Lowres=4) => 2^(2-XBIOS(4))

Z_raster%()
(0)      Zielraster-Adresse (muß gerade sein).
(1)      Zielraster-Breite in Punkten (durch 16 teilbar!).
(2)      Zielraster-Höhe in Punkten (Zeilenanzahl).
(3)      Zielraster-Breite in Words (immer: Z_raster(1)/16).
(4)      Zielraster-Format (immer 0).
(5)      Anzahl der Zielraster-Bit-Planes.
         (Hires=1/Midres=2/Lowres=4) => 2^(2-XBIOS(4))
```

R_def%()
(0) X-Koordinate des Quellrechtecks links/oben.
(1) Y-Koordinate des Quellrechtecks links/oben.
(2) X-Koordinate des Quellrechtecks rechts/unten.
(3) Y-Koordinate des Quellrechtecks rechts/unten.
(4) X-Koordinate des Zielrechtecks links/oben.
(5) Y-Koordinate des Zielrechtecks links/oben.
(6) X-Koordinate des Zielrechtecks rechts/unten.
(7) Y-Koordinate des Zielrechtecks rechts/unten.
(8) Verknüpfungsmodus (siehe PUT).

Die beiden in R_def%() beschriebenen Rechtecke sollten gleiche Größe haben, um ein nachvollziehbares Ergebnis zu liefern. Die Größe der beiden Raster kann unterschiedlich sein, wobei jedoch zu beachten ist, daß dann immer in der Größe des Quellrasters verknüpft wird (Vorsicht!).

Zum Experimentieren ist es ratsam, sich auf Operationen innerhalb des Bildschirms oder festgelegte Speicherbereiche zu beschränken, da es nicht leicht ist, die Ergebnisse dieses Rasterbefehls (bzw. der Verknüpfungsmodi) vorherzusagen und somit Fehlfunktionen oder Abstürze des Systems auftreten können.

Beispiel zum VDI-BITBLT

Da ich den in V3.0 verfügbaren RC_COPY-Befehl für sehr nützlich halte, habe ich ihn hier als BITBLT-Prozedur für die V2.xx-Versionen "nachgebaut". Die Syntax des Prozedur-Aufrufs ist mit der des RC_COPY-Befehls fast identisch. Schauen Sie sich dazu bitte das Beispiel zu RC_COPY an, wo ich die V2.xx-Aufrufe vorsorglich gleich mit eingebunden habe. Die drei winzigen Unterschiede im Aufruf sind, daß erstens statt der TO-Komponente bei RC_COPY hier ein Komma einzusetzen ist, daß zweitens die Angabe des Modus hier nicht optional ist und daß drittens die Parameterliste in Klammern zu setzen sind. Der einzige große Unterschied liegt natürlich in der Geschwindigkeit. RC_COPY ist um ein Vielfaches schneller als das V2.xx-Rc_copy.

```
' Aufruf-Beispiele siehe unter RC_COPY
'
Procedure Rc_copy(Q_ad%,X1%,Y1%,Br%,Ho%,Z_ad%,X2%,Y2%,Md%)
  Local Xt%,Yt%                  ! Lokale Variablen
  Dim A%(5),B%(5),C%(8)          ! DIM Parameterfelder
  Xt%=2-Sgn(Xbios(4))            ! X-Auflösungsteiler
  Yt%=Min(2,3-Xbios(4))          ! Y-Auflösungsteiler
  '
  ' MFDB (MemoryFormDefinitionBlock) für das Quellraster
  '----------------------------------------------------
  A%(0)=Q_ad%                    ! 1. Element = Quelladresse
```

```
    A%(1)=640/Xt%              ! 2.    "     = Rasterbreite
    A%(2)=400/Yt%              ! 3.    "     = Rasterhöhe
    A%(3)=A%(1)/16             ! 4.    "     = Breite in Words
    A%(4)=0                    ! 5.    "     = immer 0
    A%(5)=2^(2-Xbios(4))       ! 6.    "     = Bit-Planes
    '
    ' MFDB (MemoryFormDefinitionBlock) für das Zielraster
    '---------------------------------------------------
    B%(0)=Z_ad%                ! 1. Element = Zieladresse
    B%(1)=640/Xt%              ! 2.    "     = Rasterbreite
    B%(2)=400/Yt%              ! 3.    "     = Rasterhöhe
    B%(3)=B%(1)/16             ! 4.    "     = Breite in Words
    B%(4)=0                    ! 5.    "     = immer 0
    B%(5)=2^(2-Xbios(4))       ! 6.    "     = Bit-Planes
    '
    ' Rechteck-Definitionen für beide Raster
    '---------------------------------------------------
    C%(0)=X1%      ! 1. Element = Quell-X-Koord. links/oben
    C%(1)=Y1%      ! 2. Element = Quell-Y-Koord. links/oben
    C%(2)=X1%+Br%  ! 3. Element = Quell-X-Koord. rechts/unten
    C%(3)=Y1%+Ho%  ! 4. Element = Quell-Y-Koord. rechts/unten
    C%(4)=X2%      ! 5. Element = Ziel-X-Koord. links/oben
    C%(5)=Y2%      ! 6. Element = Ziel-Y-Koord. links/oben
    C%(6)=X2%+Br%  ! 7. Element = Ziel-X-Koord. rechts/unten
    C%(7)=Y2%+Ho%  ! 8. Element = Ziel-Y-Koord. rechts/unten
    C%(8)=Md%      ! 9. Element = Verknüpfungsmodus (siehe PUT)
    '
    Bitblt A%(),B%(),C%()      ! BITBLT ausführen
    Erase A%()                 !--.
    Erase B%()                    ! Felder löschen
    Erase C%()                 !--'
  Return
```

Zu Syntax-Variante 2 (nur für V3.0)

Blockadresse% ist die Startadresse eines 76 Byte großen Speicherbereichs, welcher der Reihe nach die benötigten Parameter enthält. In der folgenden Tabelle finden Sie unter Offset die Byte-Offsets der einzelnen Parameter zu dieser Adresse verzeichnet.

Zu Syntax-Variante 3 (nur für V3.0)

Parameterfeld%() ist der Name eines 4-Byte-Integer-Arrays mit mind. 23 Elementen, die der Reihe nach die benötigten Parameter enthalten. In der folgenden Tabelle finden Sie unter Index die Indizes der einzelnen Parameter verzeichnet.

Die beiden Line-A-BITBLTs der Version V3.0 arbeiten grundsätzlich identisch. Der einzige Unterschied ist, daß von dem Parameterfeld%() intern eine Kopie angelegt wird. Der Befehl verwendet dann diese Kopie für seine Arbeit. Die Variante Blockadresse% dagegen arbeitet

direkt mit dem angegebenen Speicherblock, wodurch einige Einträge nach Abschluß verändert sind (in der Tabelle mit * markiert).

Line-A-BITBLT-Parameter (nur für V3.0):

Name		Index	Offset	Bedeutung
B_WD		0	0	Rasterbreite in Pixel
B_HT		1	2	Rasterhöhe in Pixel
PLANE_CT	*	2	4	Anzahl der Farb-Ebenen
FG_COL	*	3	6	Vordergrundfarbe
BG_COL	*	4	8	Hintergrundfarbe
OP_TAB		5	10	Verknüpfungs-Maske für Vorder- und Hintergrund
S_XMIN		6	14	X-Offset im Quellraster
S_YMIN		7	16	Y-Offset im Quellraster
S_FORM		8	18	Quellraster-Adresse
S_NXWD		9	22	Quell-Offset zum nächsten Word der gleichen Ebene
S_NXLN		10	24	Quell-Offset zur nächsten Zeile der gleichen Ebene
S_NXPL		11	26	Quell-Offset zur nächsten Farbebene
D_XMIN		12	28	X-Offset im Zielraster
D_YMIN		13	30	Y-Offset im Zielraster
D_FORM		14	32	Zielraster-Adresse
D_NXWD		15	36	Ziel-Offset zum nächsten Word der gleichen Ebene
S_NXLN		16	38	Ziel-Offset zur nächsten Zeile der gleichen Ebene
S_NXPL		17	40	Ziel-Offset zur nächsten Farbebene
P_ADDR		18	42	Füllmuster-Adresse
P_NXLN		19	46	Offset zur nächsten Maskenzeile
P_NXPL		20	48	Offset zur nächsten Musterfarbe
P_MASK		21	50	Muster-Maske (vgl. HLINE)
SPACE	*	22	52-75	24-Byte-Arbeitspuffer für Blitter

Nach vielen, vielen Versuchen, einen dieser beiden Line-A-BITBLTs in Gang zu bringen, habe ich es dann ohne Erfolg und entnervt aufgegeben. Da mir der Interpreter auch partout nicht ausführlich erklären wollte, was ich den nun so sträflich falsch gemacht hatte, habe ich mich dann schmollend zurückgezogen und mich entschlossen, meinen Ärger mit Ihnen zu teilen. Ich wünsche Ihnen ein ausgeprägtes Nervenkostüm bei dem verzweifelten Versuch, diesem Befehl seinen Zauber zu entlocken.

BMOVE { BM } Speicherblock kopieren

In V3.0: { B }

BMOVE Quelle,Ziel,Anz

Ab der Adresse Quelle werden Anz Bytes gelesen und an den mit der Adresse Ziel beginnenden Bereich kopiert. Quell- und Zielbereich können sich dabei auch überschneiden.

Bei Anz ist darauf zu achten, daß nie der Wert 0 auftreten darf. Wird als Anzahl der zu kopierenden Bytes keine Konstante verwendet (z.B. ...,X_bytes*Zeilen oder ...{L~A-40}*{L~A-46}), so kann es unter ungünstigen Umständen dazu kommen, daß als Anz-Parameter Null übergeben wird. In solchen Fällen ist meistens ein gnadenloser Absturz die Folge. Um dies zu vermeiden, bietet sich die Funktion MAX(1,Anz) an, die dafür sorgt, daß mindestens ein Byte übertragen wird.

BMOVE arbeitet bei geraden Adressen schneller als bei ungeraden. Beide Adressen dürfen nicht im Supervisor-Bereich (<2048) liegen. Quell darf jedoch auch im ROM liegen.

BMOVE ist ein Befehl, ohne den professionelles Programmieren fast undenkbar wäre. Bei älteren BASIC-Dialekten bestand ein wesentlicher Unterschied zu den maschinennahen Sprachen bzw. der Maschinensprache darin, daß kein schneller Speicherblock-Transfer möglich war. Speicherblöcke mußten durch FOR..NEXT-Schleifen und PEEK/POKE Byte für Byte übertragen werden. BMOVE macht dem ein Ende. Das Einsatzgebiet ist so weit gefächert, daß es der Phantasie des Einzelnen überlassen bleibt, was er mit diesem mächtigen Befehl anfängt. An Ideen dürfte es da allerdings nicht mangeln.

11.2.1 Flimmerfreie Grafik

Beispiel 1 (Hires/Midres)

Daß "echte" Spiele in BASIC nicht programmiert werden können, war vor nur zwei Jahren noch selbstverständlich. Die neuen Dialekte - allen voran GFA-BASIC - sind dermaßen luxuriös und gewaltig, daß sich immer mehr von den "alten Assemblerhasen" dazu entschließen, nur noch die Programmteile in Assembler oder C zu schreiben, die einen wesentlichen Geschwindigkeitsvorteil bieten.

Diese Maschinensprachmodule werden dann mit C:(), CALL oder in V3.0 mit RCALL in das BASIC-Programm eingebunden. Alles andere (Windows, Menues, Strukturbefehle etc.) wird in GFA-BASIC programmiert. Warum sollte man das Rad auch zweimal erfinden? Wer sich mit Assembler oder C auskennt, weiß, welche Sysiphus-Arbeit es ist, das GEM zu den - in GFA-BASIC selbstverständlichen - kleinen Dienstleistungen zu überreden.

Das folgende Beispiel soll demonstrieren, daß auch Action-Spiele in GFA-BASIC realisiert werden können. Es ist - wohlgemerkt - als Beispiel gedacht und erfüllt keine höheren Ansprüche. Mit etwas Phantasie ließe sich dieses Programm z.B. leicht zu einem Racing-Spiel erweitern. Das kleine Auto wird übrigens mit der Maus gesteuert, wobei Vorwärtsbewegungen den Wagen beschleunigen und Rückwärtsbewegungen ihn abbremsen bzw. rückwärts fahren lassen.

```
On break cont             ! Abbruch unterbinden, damit nach Ende
'                         ! der Speicher restauriert wird (gilt
'                         ! für V2.0).
Yt%=Min(2,3-Xbios(4))     ! Y-Auflösungsteiler
@Screen(1)                ! 2. Bildschirmspeicher anlegen
' Diese Prozedur finden Sie im Anschluß an das zweite BMOVE-Beispiel.
Reserve 1000              ! BASIC-Speicher verringern
Box 1,1,11,17/Yt%         ! Auto..
Box 2,7/Yt%,10,14/Yt%     ! ..zeichnen
Get 0,0,12,18/Yt%,Auto$ ! Auto speichern
Start%=Himem+10000        ! Startadresse d. freien Speichers
' Das Offset von 10000 Byte ist hauptsächlich für die
' V2.02-Version gedacht. Die anderen Versionen können als
' Startadresse direkt HIMEM ohne Offset angeben.
Deffill ,2,3              ! DEFFILL mittelgrau
Pbox -1,-1,640,400/Yt% ! Bildschirm grau
Color 0                   ! Linienfarbe weiß
Defline ,30,2,2           ! Dicke Linie mit runden Enden
Do
  Read X%,Y%              ! Stadtplan-Koordinaten lesen
  Exit if X%=-1 And Y%=-2 ! Exit, wenn keine DATAs mehr
  If X%=-1 And Y%=-1      ! Beginn einer Strasse?
    Read X%,Y%            ! Startpunkt lesen...
    Plot X%*20,10+Y%*20/Yt% ! ...und zeichnen
  Else                    ! Eck-Koordinate?
    Draw  To X%*20,10+Y%*20/Yt% ! Dann Strasse zeichnen
  Endif
Loop
Data -1,-1,12,0,12,2,10,4,10,6,9,7,6,9,4,9,2,7,2,6,4,4,7,4,9,2
Data 10,2,12,0,-1,-1,4,4,4,2,6,0,-1,-1,10,12,10,15,12,17,12,19
Data -1,-1,4,9,1,12,1,17,-1,-1,6,19,6,17,4,15,4,13,6,12,10,12
Data 12,10,12,8,14,6,14,4,12,2,12,0,-1,-1,12,10,12,15,14,17,16
Data 17,18,15,18,12,17,9,14,8,14,6,-1,-1,28,2,28,0,-1,-1,12,10
Data 14,12,14,14,-1,-1,18,15,21,18,-1,-1,17,9,19,7,19,5,21,3,21
Data 0,-1,-1,17,9,20,9,22,10,22,12,23,13,23,16,21,18,21,19,-1,-1
Data 21,3,23,5,23,8,25,10,27,10,30,7,30,4,28,2,26,2,23,5,-1,-1
Data 27,10,31,14,31,16,28,19,-1,-1,23,13,26,13,28,15,28,19,-1,-2
Bmove Xbios(3),Start%,32000       ! 1. Screen in Puffer
```

```
Bmove Xbios(3),Start%+32000,32000 ! 2.   "   dahinter
Cls                       ! Bildschirm löschen
Hidem                     ! Maus aus
Y=225/Yt%                 ! Auto-Y-Startkoordinate
Dpoke 9952,236            ! Maus auf X-Start setzen
Repeat
  X=Mousex                ! Maus-X-Koordinate holen
  Cnt=(Cnt+Stp) Mod 400/Yt% ! Startzeile des Plan-Ausschnitts
  If Cnt<0                ! Oberhalb der Screen?
    Cnt=(399/Yt%)+Cnt     ! Dann unten wieder anfangen
  Endif
  If Peek(3584) And 224 ! Maus nach oben bewegt?
    Y=Max(120/Yt%,Y-1)    ! Neue Y-Position berechnen
    If Stp>0              ! Wagen fährt vorwärts?
      Sub Stp,0.2         ! Beschleunigungsfaktor 0.2
    Else                  ! Wagen fährt rückwärts?
      Sub Stp,0.1         ! Bremsfaktor 0.1
    Endif
  Endif
  If Peek(3584) And (Peek(3584)<8) ! Maus nach unten?
    Y=Min(280/Yt%,Y+1)    ! Neue Y-Position berechnen
    If Stp<0              ! Wagen fährt rückwärts?
      Add Stp,0.2         ! Beschleunigungsfaktor 0.2
    Else                  ! Wagen fährt vorwärts?
      Add Stp,0.1         ! Bremsfaktor 0.1
    Endif
  Endif
  Poke 3584,0             ! Mausbewegungs-Vektor löschen
  '                       ! (siehe unter ABSOLUTE)
  Bmove Start%+Int(Cnt)*80,Xbios(3)+8000,16000
  ' Bildausschnitt aus dem Puffer in die Logscreen holen
  Put X,Y,Auto$           ! Auto darüber setzen
  @Screen(2)              ! Logscreen in die Physscreen holen
Until Inkey$>"" Or Mousek ! Irgendeine Taste gedrückt?
@Screen(0)                ! Screens restaurieren
Reserve Xbios(2)-Himem-16384+Fre(0) ! Speicher restaurieren
```

Beispiel 2

Wer sieht es nicht gerne, daß sich der Bildschirm "butterweich" nach oben bewegt, wenn eine PRINT-Zeile ausgegeben wird. Man nennt das "Softscrolling" und Programme, die diesen Effekt elegant einsetzen, gewinnen dadurch erfahrungsgemäß erheblich an Publizität.

In diesem Beispiel werden innerhalb des GFA-Interpreters oder eines GFA-Compilats alle RTS gesucht und ausgegeben. Das RTS (Return from Subroutine) ist eine Rücksprunganweisung in Maschinensprache und seine Position innerhalb eines Programms für manche Assembler-Tüftler von Interesse.

```
On break cont             ! Abbruch unterbinden
' Gilt nur für die V2.0-Version, da diese Logscreen
' und Physscreen bei Programmende noch nicht selbstständig
' restauriert.
```

```
@Screen(1)               ! 2. Bildschirmspeicher anlegen
Print "*** SOFT - PRINTING *** (Moment, bitte)"
Line 0,16,639,16
@Screen(2)               ! Logscreen/Physscreen tauschen
Bp%=Basepage
@Find(0,Chr$(&H4E),Chr$(&H75),Bp%,Lpeek(Bp%+24),*Back$)
'                        ! Alle &H4E75-Words (RTS) suchen
@Screen(2)               ! Logscreen/Physscreen tauschen
Print At(1,1);"*** SOFT - PRINTING *** (<Taste>=Abbruch)"
@Screen(2)               ! Logscreen/Physscreen tauschen
Xb2%=Xbios(2)            ! Physscreen-Start holen
Xb3%=Xbios(3)            ! Logscreen-Start holen
Zeile%=17                ! Unterste Zeile der Aussparung
For I%=1 To Len(Back$)-1 Step 4 ! Nacheinander alle Longs
  '                      ! Aus dem Find-Rückgabe-String
  Exit if Back$="" Or Inkey$>"" Or Mousek ! Taste?
  Offset%=Cvl(Mid$(Back$,I%,4))-Basepage  ! Offset zur BASEPAGE
  Print At(3,1);"BASEPAGE + ";Offset%;"  <--- $4E75 (RTS)"
  For J=0 To 15          ! 16 Scan-Lines
    Bmove Xb2%+Zeile%*80+80,Xb2%+Zeile%*80,32000-Zeile%*80-80
    ' Physscreen 'hochsetzen'
    Bmove Xb3%+J*80,Xb2%+31920,80 !
    ' Logscreen-Zeile in Physscreen unten einsetzen
  Next J%
Next I%
@Screen(0)               ! Screens restaurieren
'
Procedure Find(F.flg%,Str1$,Str2$,St%,En%,F.adr_%)
  ' Sucht einen beliebigen Text im Speicher. Der Suchtext ist
  ' vorher an einer beliebigen Stelle in zwei Teile zu teilen
  ' und dann in diesen zwei Teilen zu übergeben, damit er sich
  ' nicht selbst finden kann.
  '
  ' F.flg% = Flag
  '          0   = Der angegebene Bereich wird in seiner
  '                gesamten Länge durchsucht und alle
  '                gefundenen Positionen zurückgegeben.
  '          <>0 = Die Suche wird nach dem ersten gefundenen
  '                Ausdruck abgebrochen.
  ' Str1$  = Vorderer Teil des Suchtextes
  ' Str2$  = Hinterer Teil des Suchtextes
  ' St%    = Startadresse des zu durchsuchenden Bereichs
  ' En%    = Endadresse des zu durchsuchenden Bereichs.
  '          Beide Adressen werden ggfs. automatisch auf
  '          die nächstkleinere gerade Adresse "gerundet".
  '          Kleinere Adressen als 2048 werden automatisch
  '          auf 2048 hochgesetzt. Nach oben gibt es keine
  '          Begrenzung, um ggfs. auch das ROM durchsuchen
  '          zu können.
  ' F_adr% = Pointer auf eine Rückgabe-String-Variable, die
  '          nach Abschluß alle gefundenen Positionen im
  '          MKL$()-Format enthält. Die Anzahl gefundener
  '          Positionen ergibt sich also aus Len(Back$)/4.
  '
  Local F.b$,F.i%,F.of%,F.step%,F.spc%,F.sta%,F.buf$
  St%=Max(2048,Int(St%/2)*2) ! Startadresse trimmen
  En%=Max(St%,Int(En%/2)*2)  ! Endadresse trimmen
  F.step%=Min(En%-St%,4000)  ! Suchblock-Größe
  F.rst%=(En%-St%) Mod F.step% ! Block-Rest berechnen
```

```
    F.b$=Space$(F.step%)        ! Suchpuffer einrichten
    F.spc%=F.step%              ! Kopie der Block-Größe
    For F.i%=St% To En% Step F.step% ! Bereich durchlaufen
      If (F.i%+F.step%)>En%     ! Rest kleiner als Blockgröße?
        F.spc%=F.rst%           ! Dann Blockgröße gleich Rest
      Endif
      Bmove F.i%,Varptr(F.b$),Max(F.spc%,1) ! Block...
      '                         ! ...in den Puffer kopieren
      Clr F.sta%                ! INSTR-Startposition =0
      Repeat
        F.of%=Instr(Left$(F.b$,F.spc%),Str1$+Str2$,F.sta%)
        '                       ! Such-String gefunden?
        If F.of%                ! Wenn ja,
          F.sta%=F.of%+Len(Str1$+Str2$) ! dann neue
          '                     ! INSTR-Startposition setzen
          F.buf$=F.buf$+Mkl$(F.i%+F.of%-1) ! und Adresse
        Endif                   ! in MKL$-String einbinden
      Until F.of%=0 Or (F.flg%<>0 And F.of%>0) ! Abbruch, wenn nichts
      ' gefunden wurde oder nur die 1. Position gesucht wird
§     Exit if F.flg%              ! Abbruch, wenn F.flg gesetzt
      Exit if F.flg% And F.of%>0 ! Abbruch, wenn F.flg gesetzt
    Next F.i%                     ! Nächsten Block holen
    *F.adr_%=F.buf$               ! MKL$()-String zurückgeben
  Return
  '
  '   Achtung: Im Programm FIND.LST auf der Diskette müssen zwei
  '   fehlerhafte Zeilen ausgetauscht werden. Es handelt sich um
  '   die vierte vor "Return", die mit "Exit..." beginnt und die
  '   sechste vor "Return", welche mit "Until..." beginnt.
  '
  Procedure Screen(Flag%)
    '
    ' Dies ist eigentlich die zentrale Prozedur zur Erzeugung
    ' von flimmerfreien, bewegten Bildern. Sie legt einen zweiten
    ' Speicher für die Logscreen (siehe XBIOS(5)/XBIOS(3)) an und
    ' man hat dann die Möglichkeit, die innerhalb des Speichers
    ' - unsichtbar - erzeugte Grafik in rasanter Geschwindgkeit
    ' auf einmal in den sichtbaren Bildschirm-Speicher (Physbase
    ' siehe XBIOS(2)/XBIOS(5)) zu transferieren. Der vorher auf der
    ' Physscreen liegende - sichtbare - Bildschirm-Inhalt liegt
    ' anschließend in der unsichtbaren Logscreen. Dabei werden nicht
    ' die Speicherinhalte vertauscht, sondern dem Video-Shifter
    ' (Arbeits-Chip zur Übertragung eines 32-KByte-Speichers auf
    ' den Monitor) wird einfach eine andere Startadresse des
    ' darzustellenden Speichers übergeben.
    '
    ' Flag% = 1
    '    Es wird ein Integerfeld als Buffer für die Logscreen
    '    eingerichtet und die Adressen der Screens in Pbase%
    '    und Lbase% abgelegt.
    ' Flag% = 2
    '    Hiermit werden die beiden Screens umdefiniert. Die
    '    physikalische Screen (der auf dem Monitor
    '    sichtbare Bildschirmspeicher) wird zur logischen Screen
    '    (der für Ausgaben relevante Bildschirmspeicher) und
    '    umgekehrt. D.h., daß während etwas auf der Logscreen
    '    ausgegeben wird, dieses auf der Physscreen nicht zu
    '    sehen ist. Die Ausgaben werden erst sichtbar, wenn man
    '    das nächste Mal Screen(2) aufruft. Wurde vorher kein
```

```
'    Screen(1)-Aufruf durchgeführt, bleibt Screen(2)
'    ohne Wirkung.
' Flag% = 3
'    Restauriert wieder den ursprünglichen Zustand. Beide
'    Screens werden wieder auf dieselbe Adresse gelegt.
'    Dadurch sind Bildschirmausgaben wieder direkt sichtbar.
'    Der für die alte Logscreen reservierte Speicher wird
'    wieder gelöscht. Die Logscreen ist danach für den
'    nächsten Einsatz erst wieder durch Screen(1) zu
'    initialisieren. Die beiden globalen Variablen Lbase%
'    und Pbase% dürfen zwischen dem Screen(1)- und dem
'    entsprechenden Screen(0)-Aufruf nicht verändert werden,
'    da sonst Absturz droht.
'    Ab GFA-BASIC V2.02 werden die Screens bei Programmende
'    vom BASIC automatisch wieder restauriert.
'
If Flag%=1              ! Logscreen einrichten?
  Dim L_adr%(8064)      ! DIM Puffer-Feld
  Pbase%=Xbios(2)       ! Physscreen-Adresse
  Lbase%=Int(Varptr(L_adr%(0))/256+1)*256 ! Neue
Endif                   ! Logscreen-Adresse
If (Flag%=2) And Dim?(L_adr%()) ! Screens tauschen?
  Swap Lbase%,Pbase%    ! Startadressen tauschen
  Void Xbios(5,L:Pbase%,L:Lbase%,-1) ! und Screens
Endif                   ! neu setzen
If (Flag%=0) And Dim?(L_adr%()) ! Screens restaurieren?
  Void Xbios(5,L:Max(Pbase%,Lbase%),L:Max(Pbase%,Lbase%),-1)
  ' Screens beide auf die höhere der beiden Adressen setzen
  Erase L_adr%()        ! Puffer-Feld löschen
Endif
Return
```

Das was Sie bis jetzt in diesem Kapitel gesehen haben, bewegt sich noch im normalen Rahmen. Bei der Grafik-Animation stellt sich immer wieder die große Frage: "Wie bekomme ich die Geometrie in den Griff?"

Geometrie ist nicht jedermanns Sache. Manchen (wie mir) macht sie geradezu "kindischen" Spaß, weil nur mit der Geometrie eine Nachahmung natürlicher Vorgänge (die Grundidee der Computer-Animation) möglich ist. In vielen Animations-Büchern und -Artikeln wird man pausenlos mit hochwissenschaftlichen Formeln traktiert, die selbst ausgefuchsten Mathematikern teilweise die Haare zu Berge stehen lassen. Ich selbst studiere Architektur, was in vielen Beziehung quasi eine Unter-Disziplin der Mathematik und Physik ist (Statik, Massenberechnung, Bauphysik, darstellende Geometrie etc.), aber bei diesen Formeln muß ich oft kleinlaut die Segel streichen. Ich nehme an, daß es den meisten von Ihnen genauso ergeht.

Um nun dem Geometrie-Ungeübten ein vielseitiges Werkzeug an die Hand zu geben, habe ich eine kleine Prozedur entwickelt, auf die ich - was Sie mir hoffentlich zugestehen werden - besonders stolz bin.

Die einzigen Grundkenntnisse, über die Sie beim Einsatz dieser Routine verfügen müssen, sind die Kenntnis über die Konstruktion eines Kreises (Pi, Sinus, Cosinus) bzw. eine Vorstellung von der umlaufenden Gradeinteilung bei einem Kreis. Was man dann mit Star - so heißt die Prozedur bedeutungsschwer - alles machen kann, wird Ihnen hoffentlich ein wenig die Sprache verschlagen. Star ist - für mich jedenfalls - der vielseitigste Grafik-Befehl, den ich kenne. Der Rest ist dann nur noch Übung.

```
On Break Cont              ! Break abfangen
Xt%=2-Sgn(Xbios(4))        ! X-Auflösungsteiler
Yt%=Min(2,3-Xbios(4))      ! Y-Auflösungsteiler
Deftext ,17,,26/Yt%
Text 100/Xt%,200/Yt%,440/Xt%,"* STAR *"
Print At(26/Xt%,16);"Alle folgenden Demonstrationen"
Print At(26/Xt%,17);"verwenden im wesentlichen ein-"
Print At(26/Xt%,18);"und denselben Grafik-Befehl!"
Pause 200
'
@Screen(1)                 ! 2. Screen einrichten (s.o.)
@Screen(2)                 ! 2. Screen initialisieren (s.o.)
'
Deffill ,2,2
For I%=0 To 180
  Cls
  @Star(2,320/Xt%,200/Yt%,100/Xt%,50/Yt%,I%,I%,320/Xt%,...
               ...200/Yt%,200/Xt%,100/Yt%,I%,I%,8,1,I%+180)
  @Star(2,320/Xt%,200/Yt%,100/Xt%,50/Yt%,I%,I%,320/Xt%,...
                ...200/Yt%,20/Xt%,10/Yt%,I%,I%,8,1,360-I%)
  @Star(1,320/Xt%,200/Yt%,100/Xt%,50/Yt%,I%,I%,320/Xt%,...
               ...200/Yt%,200/Xt%,100/Yt%,I%,I%,8,1,180-I%)
  @Star(1,320/Xt%,200/Yt%,100/Xt%,50/Yt%,I%,I%,320/Xt%,...
                   ...200/Yt%,20/Xt%,10/Yt%,I%,I%,8,1,I%)
  @Screen(2)               ! Screens tauschen
Next I%
'
For I%=200 To 0 Step -1
  Cls
  @Star(1,320/Xt%,200/Yt%,(300-I%)/Xt%,(300-I%)/Yt%,I%,I%,...
      ...320/Xt%,200/Yt%,(20+I%)/Xt%,(20+I%)/Yt%,0,0,16,0,I%)
  @Screen(2)               ! Screens tauschen
Next I%
'
For I%=1 To 500 Step 3
  Cls
  J%=J%-I%
  @Star(1,(500-I%)/Xt%,(300-I%*0.5)/Yt%,140/Xt%,140/Yt%,...
   ...I%/30,I%/30,(600-I%)/Xt%,(200+Sin(I%/180)*60)/Yt%,...
                ...30+I%/20,30+I%/20,I%/30,I%/30,4,0,J%/4)
  @Screen(2)               ! Screens tauschen
Next I%
'
Cls
For I%=1 To 100
  Cls
  @Star(1,320/Xt%,200/Yt%,100/Xt%,50/Yt%,I%/5,I%/5,326/Xt%,...
```

```
          ...206/Yt%,(10+I%)/Xt%,(5+I%)/Yt%,I%/5,I%/5,4,0,I%*4)
  @Screen(2)                 ! Screens tauschen
Next I%
'
Cls
Deffill ,2,4
For I%=-60 To 160 Step 4
  Cls
  @Star(1,(200+I%)/Xt%,(200-I%)/Yt%,(300-I%)/Xt%,...
  ...(200-I%*1.2)/Yt%,0,0,(200+I%)/Xt%,220-I%)/Yt%,...
        ...(300-I%)/Xt%,(200-I%*1.2)/Yt%,0,0,4,0,I%/2)
  @Screen(2)                 ! Screens tauschen
Next I%
'
Cls
Deffill ,2,1
Pellipse 320/Xt%,300/Yt%,240/Xt%,70/Yt%
Deffill ,2,4
Pellipse 320/Xt%,300/Yt%,30/Xt%,12/Yt%
Sget Aa$
For I%=30 To 620 Step 8
  Sput Aa$
  If I%>30
    Circle Px%(0),Py%(0),1
    Circle Px%(3),Py%(3),1
  Endif
  Sget Aa$
  Deffill ,2,4
  @Star(2,320/Xt%,300/Yt%,240/Xt%,70/Yt%,0,0,320/Xt%,...
                  ...300/Yt%,30/Xt%,12/Yt%,0,0,6,1,I%)
  @Star(2,320/Xt%,300/Yt%,240/Xt%,70/Yt%,0,0,320/Xt%,...
              ...300/Yt%,30/Xt%,12/Yt%,0,0,6,1,I%+180)
  Deffill ,2,3
  @Star(1,320/Xt%,300/Yt%,240/Xt%,70/Yt%,0,0,320/Xt%,...
            ...(80+I%/4)/Yt%,30/Xt%,12/Yt%,0,0,6,1,I%)
  @Star(1,320/Xt%,300/Yt%,240/Xt%,70/Yt%,0,0,320/Xt%,...
         ...(80+I%/4)/Yt%,30/Xt%,12/Yt%,0,0,6,1,I%+180)
  Pellipse 320/Xt%,80/Yt%,30/Xt%,12/Yt%
  @Screen(2)                 ! Screens tauschen
Next I%
Bmove Xbios(2),Xbios(3),32000
Print "MOMENT, BITTE !"
@Screen(2)                 ! Screens tauschen
'
Deffill ,2,8
Pbox 0,0,639/Xt%,399/Yt%
Dpoke Vdibase+34,0         ! für V2.xx
'    Boundary 0            ! für V3.0
Deffill ,2,4
For I%=1 To 360 Step 12
  Add Ii,0.55
  Deffill ,2,Abs(9-Ii)
  @Star(2,500/Xt%,200/Yt%,40/Xt%,40/Yt%,0,0,600/Xt%,...
              ...40/Yt%,20/Xt%,20/Yt%,0,0,30,1,I%+45)
Next I%
Ii=0
For I%=1 To 360 Step 12
  Add Ii,0.4
  Deffill ,2,Abs(8-Ii)
```

```
  @Star(2,300/Xt%,140/Yt%,90/Xt%,90/Yt%,0,0,500/Xt%,...
               ...200/Yt%,40/Xt%,40/Yt%,0,0,30,1,I%+80)
Next I%
Circle 300/Xt%,140/Yt%,90/Yt%
Ii=0
For I%=0 To 356 Step 12
  Add Ii,0.33
  Deffill ,2,Abs(-1-Ii) Mod 8
  @Star(2,150/Xt%,280/Yt%,200/Xt%,200/Yt%,0,0,300/Xt%,...
                    ...140/Yt%,90/Xt%,90/Yt%,0,0,30,1,I%)
Next I%
Circle 150/Xt%,280/Yt%,200/Yt%
Dpoke Vdibase+34,1          ! für V2.xx
'    Boundary 1             ! für V3.0
Sget Aa$
For I%=1 To 166 Step 3
  Cls
  Sput Aa$
  @Star(2,(150+I%)/Xt%,(280-I%/2)/Yt%,(160-I%)/Xt%,...
             ...(140-I%)/Yt%,-30,-30,(266+I%/3)/Xt%,...
                     ...(210-I%/8)/Yt%,0,0,0,0,4,1,45)
  @Screen(2)                  ! Screens tauschen
Next I%
Cls
@Screen(2)                  ! Screens tauschen
Cls
'
Pbox 10/Xt%,10/Yt%,629/Xt%,389/Yt%
Deffill ,0,0
For I%=1 To 7
  Deffill ,2,I%
  Pcircle 320/Xt%,200/Yt%,(200-I%*25)/Yt%
Next I%
Sget Aa$
@Screen(2)                  ! Screens tauschen
For I%=1 To 640 Step 8
  Cls
  Sput Aa$
  Deffill ,2,8
  @Star(2,I%/Xt%,(200+Sin(I%/180)*120)/Yt%,200/Xt%,200/Yt%,...
                ...22,22,I%/Xt%,(200+Sin(I%/180)*120)/Yt%,...
                                  ...20/Xt%,20/Yt%,0,0,4,1,I%)
  Deffill ,0,0
  @Star(2,I%/Xt%,(206+Sin(I%/180)*120)/Yt%,200/Xt%,200/Yt%,...
                ...22,22,I%/Xt%,(206+Sin(I%/180)*120)/Yt%,...
                                  ...20/Xt%,20/Yt%,0,0,4,1,I%)
  Deffill ,2,8
  @Star(2,I%/Xt%,(200+Sin(I%/180)*120)/Yt%,180/Xt%,180/Yt%,...
                ...40,40,I%/Xt%,(200+Sin(I%/180)*120)/Yt%,...
                                ...80/Xt%,80/Yt%,30,30,4,1,I%)
  Deffill ,2,4
  @Star(2,I%/Xt%,(210+Sin(I%/180)*120)/Yt%,180/Xt%,180/Yt%,...
                ...40,40,I%/Xt%,(210+Sin(I%/180)*120)/Yt%,...
                                ...80/Xt%,80/Yt%,30,30,4,1,I%)
  @Screen(2)                  ! Screens tauschen
Next I%
Pause 10
Cls
'
```

```
Deffill ,2,8
Pbox 0,0,639/Xt%,399/Yt%
Deffill ,0,0
Pcircle 320/Xt%,200/Yt%,199/Yt%
Circle 320/Xt%,200/Yt%,195/Yt%
Circle 320/Xt%,200/Yt%,146/Yt%
Deffill ,3,9
Pcircle 320/Xt%,200/Yt%,140/Yt%
Deffill ,2,4
Pcircle 320/Xt%,200/Yt%,20/Yt%
Sget Aa$
For I%=0 To 360 Step 6
  Repeat
  Until (Timer-T)/200=>1
  Sput Aa$
  @Star(2,320/Xt%,200/Yt%,190/Xt%,190/Yt%,-2,9,320/Xt%,...
              ...200/Yt%,160/Xt%,160/Yt%,-2,9,30,1,I%-90)
  @Star(1,320/Xt%,200/Yt%,100/Xt%,100/Yt%,0,12,320/Xt%,...
              ...200/Yt%,30/Xt%,30/Yt%,0,12,30,1,I%-90)
  Print At(66/Xt%,2);"00:00:";Spc(2-Len(Str$(I%/6)));I%/6
  If I% Mod 30=0
    Pcircle Px%(1),Py%(1),6/Yt%
    Graphmode 3
    @Star(2,320/Xt%,200/Yt%,190/Xt%,190/Yt%,-2,9,320/Xt%,...
              ...200/Yt%,160/Xt%,160/Yt%,-2,9,30,1,I%-90)
    Graphmode 1
    @Star(1,320/Xt%,200/Yt%,190/Xt%,190/Yt%,-2,9,320/Xt%,...
              ...200/Yt%,160/Xt%,160/Yt%,-2,9,30,1,I%-90)
    Sget Aa$
  Endif
  T=Timer
  @Screen(2)              ! Screens tauschen
Next I%
Pause 100
'
@Screen(0)
'
Procedure Star(Md%,Xpa%,Ypa%,Xra%,Yra%,Oa1,Oa2,Xpi%,Ypi%,...
                    ...Xri%,Yri%,Oi1,Oi2,Eck%,Pkt%,Sw)
  ' Zeichnet ein beliebiges Vieleck innerhalb zweier Radien.
  '
  ' Md% = Modus     1 = Figur als POLYLINE
  '                 2 = Figur als POLYFILL
  '                 3 = Figur als POLYMARK
  ' Xpa%= X-Koord. des Mittelpunktes d. äußeren Kreises/Ellipse
  ' Ypa%= Y-Koord. des Mittelpunktes d. äußeren Kreises/Ellipse
  ' Xra%= X-Radius des äußeren Kreises/Ellipse
  ' Ypa%= Y-Radius des äußeren Kreises/Ellipse
  ' Oa1 = Offset (in Grad) der vorderen Flächen-Außenkoordinate
  ' Oa2 = Offset (in Grad) der hinteren Flächen-Außenkoordinate
  '
  ' Xpi%= X-Koord. des Mittelpunktes d. inneren Kreises/Ellipse
  ' Ypi%= Y-Koord. des Mittelpunktes d. inneren Kreises/Ellipse
  ' Xri%= X-Radius des inneren Kreise/ellipse
  ' Ypi%= Y-Radius des inneren Kreise/Ellipse
  ' Oi1 = Offset (in Grad) der vorderen Flächen-Innenkoordinate
  ' Oi2 = Offset (in Grad) der hinteren Flächen-Innenkoordinate
  '
  ' Eck%= Anzahl der absoluten Teilflächen (beliebig)
```

```
' Pkt%= Anzahl der zu zeichnenden Teilflächen
'        0 = Figur komplett zeichnen
'        n = beliebige Punktanzahl (max. Eck%)
' Sw  = Startwinkel in Grad
'       Null = rechter Endpunkt der Horizontalachse.
'       Die vorderen Flächenkoordinaten werden bei Sw
'       (ggfs. +/- Offset) gezeichnet.
'       Alle weiteren folgen im Uhrzeigersinn
'
' Nach Rückkehr zum Programm können die Flächenkoordinaten der
' zuletzt gezeichneten Fläche ausgelesen werden:
'
'   Px%(0)/Py%(0) = Vordere Koordinate im Innen-Kreis/Ellipse
'   Px%(1)/Py%(1) = Vordere Koordinate im Aussen-Kreis/Ellipse
'   Px%(2)/Py%(2) = Hintere Koordinate im Aussen-Kreis/Ellipse
'   Px%(3)/Py%(3) = Hintere Koordinate im Innen-Kreis/Ellipse
'
Local J%,Stp,I
Eck%=Max(1,Abs(Eck%))       ! Mind. eine Ecke
If Pkt%=0                   ! Punkteanzahl = 0?
  Pkt%=Eck%                 ! Punkteanzahl = Eckanzahl
Else                        ! Punkte <> 0 !
  Pkt%=Min(Eck%,Abs(Pkt%)) ! Punkteanzahl = max. Ecken
Endif
Stp=360/(Eck%+1.0E-100) ! Winkelschritte
Erase Px%()
Erase Py%()
Dim Px%(4),Py%(4)
For I=90-Sw To -270-Sw Step -Stp          ! Einmal rundum
  Px%(0)=Xpi%+Sin((I-Oi1)*Pi/180)*Xri%    !------.
  Py%(0)=Ypi%+Cos((I-Oi1)*Pi/180)*Yri%    !      |
  Px%(1)=Xpa%+Sin((I-Oa1)*Pi/180)*Xra%    !      |
  Py%(1)=Ypa%+Cos((I-Oa1)*Pi/180)*Yra%    !      |      Flächen-
  Px%(2)=Xpa%+Sin((I-Stp+Oa2)*Pi/180)*Xra% !     |---  punkte
  Py%(2)=Ypa%+Cos((I-Stp+Oa2)*Pi/180)*Yra% !     |     berechnen
  Px%(3)=Xpi%+Sin((I-Stp+Oi2)*Pi/180)*Xri% !     |
  Py%(3)=Ypi%+Cos((I-Stp+Oi2)*Pi/180)*Yri% !-----'
  Px%(4)=Px%(0)
  Py%(4)=Py%(0)
  If Md%=1                  ! Modus = 1?
    Polyline 5,Px%(),Py%() ! Dann POLYLINE
  Endif
  If Md%=2                  ! Modus = 2?
    Polyfill 4,Px%(),Py%() ! Dann POLYFILL
  Endif
  If Md%=3                  ! Modus = 3?
    Polymark 4,Px%(),Py%() ! Dann POLYMARK
  Endif
  Inc J%                    ! Punktezähler erhöhen
  Exit if J%=Pkt%           ! Abbruch, wenn Punkteanzahl erreicht
 Next I                     ! Nächste Fläche
Return
```

Version 3.0

BYTE{}, CARD{}, LONG{} Speicherinh. lesen (User-Modus)
BYTE{}= {BY}= } 1 Byte schreiben (User-Modus)
CARD{}= {CA}= } 2 Byte schreiben (User-Modus)
LONG{}= {LON}= } 4 Byte schreiben (User-Modus)

Syntax (Zuweisung an Adresse/vgl. D-L-POKE):
BYTE{Adresse}=Wert => Schreibt ein Byte
CARD{gerade Adresse}=Wert => Schreibt zwei Byte (Word)
LONG{gerade Adresse}=Wert => Schreibt vier Byte (Long)
Syntax (Lesen aus Adresse/vgl. D-L-PEEK):
Var=BYTE{Adresse} => Liest ein Byte
Var=CARD{gerade Adresse} => Liest zwei Byte (Word/(Cardinal)
Var=LONG{gerade Adresse} => Liest vier Byte (Long)

Ab Adresse werden dem Format entsprechend viele (1, 2, 4) Bytes gelesen bzw. geschrieben. Bei CARD{} und LONG{} dürfen nur gerade Adressen verwendet werden. Die Lese-Funktionen arbeiten hier - im Gegensatz zu D-L-PEEK() - generell im User-Modus. D.h, daß auf die Supervisor-Bereiche (<2048) **nicht** zugegriffen werden kann. Dadurch, daß intern keine Umschaltung in den Supervisor-Modus vorgenommen werden muß, benötigen diese Lesefunktionen nur ca. 60 Prozent der Zeit einer vergleichbaren PEEK()-Funktion.

Sollen 4-Byte-Werte gelesen werden, kann die Bezeichnung LONG weggelassen werden (z.B. {XBIOS(2)} liest 4 Byte ab XBIOS(2)).

Beispiel:

Echte System-Zeitlupe in BASIC - und das ohne einen einzigen Maschinensprache-Befehl? Wo gibt es denn so etwas? Wenn Sie auch so denken, dann lassen Sie sich überraschen - das gibt es nämlich **doch**!

```
On break gosub Break    ! Wichtig! Breaks abfangen und ggfs. vor
'                       ! Programmende den VBl-Queue restaurieren
Zeit%=476               ! Beliebige Zeitvorgabe
@Slow(0)                ! Zeitlupen-Queue initialisieren
Again:                  ! Label für Wiederholung  <------------.
@Slow(Zeit%)            ! Zeitlupe einschalten                 !
@Sc(">> SLOWMOTION - TEST <<") ! Screen zeichnen               !
Tron                    ! Zur Demonstration Trace-Modus an     !
Al$="Ist "+Str$(Zeit%)+" langsam genug?"                       !
Alert 2,Al$,1,"Aber|ja doch!|Nein",Back%                       !
If Back%=3              ! Noch langsamer?                      !
  Alert 1,"Na dann nochmal,|etwas langsamer !",1,"OKAY",Back%  !
  Add Zeit%,3           ! Zeitwert um 3 erhöhen                !
```

```
  Troff                   ! Trace-Modus aus                      !
  Goto Again              ! Nochmal von vorn    >-----------------'
Endif
Troff                     ! Trace Off
@Slow(-1)                 ! Zeitlupe ausschalten
@Sc("Wieder Normalgeschwindigkeit") ! Screen zeichnen
'
Procedure Slow(Sl.md%)
  ' Vbl-Interrupt-abhängige Zeitlupe! Verzögert die Ausführungen
  ' aller Aktionen durch Mehrfachausführung der Standard-Vbl-Routine.
  '
  ' Sl.md% = Modus bzw. Verzögerungswert:
  '           0 = Ist der Standard-Vbl-Interrupt intakt (also
  '               z. Zt. keine Zeitlupe an), wird hiermit die
  '               Zeitlupe initialisiert.
  '          -1 = Ist die Zeitlupe initialisiert, wird hiermit
  '               der Standard-Zustand wieder hergestellt.
  '               Anschließend kann SLOW mit evtl. anderen
  '               Verzögerungwerten neu aufgerufen werden, ohne
  '               vorher wieder durch SLOW(0) initialisiert
  '               werden zu müssen.
  '               SLOW muß erst dann neu intialisiert werden,
  '               wenn nach SLOW(-1) durch RESERVE die Lage
  '               von HIMEM verändert wurde. Bei Version V2.02
  '               muß statt HIMEM die Adresse HIMEM+5000 angegeben
  '               werden.
  '   1 bis 499 = Verzögerungswert:
  '               Da tatsächlich alles (!) durch diese Zeitlupe
  '               verzögert wird, verzögern sich also auch die
  '               Interruptroutinen selbst.
  '               Aus diesem Grund wirken sich die Zeitwerte
  '               quadratisch aus. D.h., daß eine effektive
  '               Zeitlupe erst ca. ab dem Wert 400 feststellbar ist
  '               und von da ab jede Erhöhung des Zeitwertes die
  '               Verzögerung potenziert.
  '               Werte bis zu 495 sind noch erträglich. Größere
  '               Werte führen zu extremer Zeitlupe, so daß die
  '               Ausführung eines einzelnen BASIC-Befehls bis
  '               zu mehreren Minuten in Anspruch nehmen kann !!!
  ' Vorsicht:
  ' Bei Disketten-Zugriffen während der Zeitlupe dürfen auf keinen
  ' Fall FILESELECT-Boxen aufgerufen werden!! -Absturz-!!
  ' Auch nicht zum Laden und Speichern von Programmen. Es sind nur
  ' dann direkte Diskettenzugriffe möglich, wenn die Zeitlupe
  ' nicht zu langsam ist, da sonst Disk-Errors entstehen.
  '
  ' Wird in Adresse $454 wieder eine 8 (SDPOKE) und in Adresse $456
  ' der Wert $4CE (SLPOKE) geschrieben, ist alles wieder so, als ob
  ' nichts gewesen wäre. Dann ist auch die FILESELECT-Box wieder
  ' verwendbar.
  '
  ' Sollte aus irgendwelchen Gründen der Prozedur-Aufruf mit 'Sl.md%'
  ' = -1 nicht möglich sein, sind die beiden eben genannten POKEs im
  ' Interpreter-Betrieb via Direktmodus auszuführen, was dann jedoch
  ' je nach vorher gültigem Sl.md%-Zeitwert einige Zeit und evtl.
  ' viel Geduld in Anspruch nimmt. In Compilaten gibt es allerdings
  ' keinen Notausgang, der in den allermeisten Fällen allerdings auch
  ' nicht notwendig sein dürfte.
  '
```

```
  ' Zusätzlich zur eingestellten Zeitlupe, kann die Geschwindigkeit
  ' der Programmausführung durch leichte Mausbewegungen
  ' weiter verlangsamt werden. Je nach Größe des Sl.md%-Zeitwertes
  ' sogar bis zum (fast) absoluten Stillstand.
  '
  ' Die Adressen sind im Supervisor-Bereich, weshalb hier
  ' auch für V3.0 SL-/SDPOKE, bzw. L/PEEK() verwendet wird.
  ' Die Routine ist auch für die V2.xx-Versionen verwendbar, wenn die
  ' CARD{}- und LONG{}-Aufrufe durch DPEEK()/DPOKE bzw. LPEEK()/LPOKE
  ' ersetzt werden.
  '
  Local J%,I%
  ' -------------------------------------------------------------.
  If Sl.md%<0 And Card{Himem+2}>0 ! Sl.md% < 0 und die Zeitlupe  !
    '                             ! ist initialisiert?           !
    Sdpoke &H454,Card{Himem+2}    ! nvbls restaurieren           !
    Slpoke &H456,&H4CE            ! Alte vbl_list in vblqueue    !
  Endif !======================================================.
  If Sl.md%=0 And Lpeek(&H456)=&H4CE ! Sl.md% = 0 und alter    !
    '                             ! vblqueue noch intakt?        !
    Reserve Xbios(2)-16384-Himem+Fre(0)-2100 ! 2100 Byte reservieren!
    Card{Himem+2}=Dpeek(&H454)    ! Alten nvbls merken           !
    For J%=0 To 7                 ! Acht...                      !
      Exit if Lpeek(&H4CE+J%*4)=0 ! ...belegte vbl_list-Plätze in !
      {Himem+4+J%*4}=Lpeek(&H4CE+J%*4) ! neue vbl_list übernehmen !
    Next J%                                                      !
    '   -----------Neue vbl_list einrichten--------------        !
    For I%=0 To 500               ! 500 mal...                   !
      {Himem+4+J%*4+I%*4}={Himem+4} ! ...Standard-Vbl-Adresse... !
    Next I%                       ! ...hintereinander kopieren   !
  Endif !======================================================.
  If Sl.md%>0 And Sl.md%<500 And {Himem+4}=Lpeek(&H4CE)         !
    ' Sl.md% > 0 und Sl.md% < 500 und Standard-Adresse           !
    '                             ! schon übertragen?            !
    Slpoke &H456,Himem+4          ! Neue Listenadresse in vblqueue !
    Sdpoke &H454,Sl.md%           ! Neue Anzahl in nvbls         !
  Endif !------------------------------------------------------'
Return
Procedure Break       ! Abfang-Routine für <Control/Shift/Alternate>
  Troff               ! Ggfs. Trace-Modus ausschalten
  If Lpeek(&H456)<>&H4CE ! Zeitlupe initialisiert?
    '                 ! Also kein Standard-Vbl-Queue?
    @Slow(-1)         ! Dann Standard-Vbl-Queue restaurieren
  Endif
  Reserve Xbios(2)-16384-Himem+Fre(0) ! BASIC-RAM restaurieren
  Edit                ! Programmende
Return
Procedure Sc(Tx$)     ! Aufbau-Routine für die Demo-Screen
  Cls                 !------------------------------.
  Text 210,13,220,Tx$                                  !
  Deffill ,2,4                                         !
  Pbox 2,16,637,397                                    !
  Deffill ,0,0                                         !- Demo-Screen
  For I%=0 To 9                                        !  zeichnen
    For J%=0 To 5                                      !
      Pbox I%*63+4,J%*62+18,I%*63-4+68,J%*62-2+76      !
    Next J%                                            !
  Next I%             !------------------------------'
Return
```

Version 3.0

CHAR{} — C-Text lesen
CHAR{}= {CH}= } — ... schreiben

Var$=CHAR{Adresse} **=> C-Text lesen (Funktion)**
CHAR{Adresse}=Expr$ **=> C-Text schreiben**

Es wird ein String im C-Format gelesen bzw. geschrieben. Bei der Schreibfunktion wird dem String Expr$ automatisch ein Null-Byte angehängt und der String an Adresse geschrieben. Die Lesefunktion liest ab Adresse, bricht beim ersten gefundenen Null-Byte ab und liefert den bis dahin gelesenen Text zurück.

Beispiele finden Sie unter anderem bei FSNEXT(), unter KEYDEF, sowie bei den AES-Library-Beispielen.

Ich fand es an CHAR{} äußerst störend, daß man "nur" mit Null-Bytes endende Strings ermitteln kann. Gerade für die freie String-Verwaltung innerhalb des freien RAMs hinter HIMEM (z.B. durch BGET# geladenen ASCII-Dateien zur Verarbeitung mit BMOVE) oder zur beliebigen CR/LF- bzw. CR- oder LF-String-Suche innerhalb des Speichers hielt ich es für angebracht, eine Prozedur zu entwickeln, die ähnlich funktioniert wie CHAR{}, aber beliebige Suchzeichen zuläßt. Die folgende Prozedur erfüllt diese Bedingungen. Dabei gibt es eine Einschränkung: Strings mit einer größeren Länge als 10000 Bytes, können nicht gefunden werden. Sie können diese maximale Länge allerdings leicht auf 32700 Bytes erweitern, indem Sie die SPACE$()- und BMOVE-Zeile entsprechend ändern. Diese Prozedur ist natürlich auch in V2.xx verwendbar.

```
DO
  ' alle Strings suchen, die im Interpreter mit
  ' einem Carriage Return enden.
  @cchar(BASEPAGE+90000+cnt%,CHR$(13),*a$)
  ADD cnt%,LEN(a$)
  PRINT a$
LOOP
PROCEDURE cchar(adr%,sign$,buff%)
  '  Adr%  = Startadresse des String. Das erste Byte des Strings ist
  '          also das Byte, welches an Adr% liegt.
  '  Sign$ = Such-String (auch mehrere Zeichen)
  '  Buff% = Pointer auf eine String-Rückgabevariable, die nach
  '          Abschluß ggfs. den extrahierten String enthält.
  '          Der Rückgabe-String schließt dann mit der Suchvorgabe ab
  LOCAL buff$,s.pos%
  buff$=SPACE$(10000)
  BMOVE adr%,VARPTR(buff$),10000
  s.pos%=INSTR(buff$,sign$)
```

```
  IF s.pos%
    *buff%=LEFT$(buff$,s.pos%-1+LEN(sign$))
  ENDIF
RETURN
```

Version 3.0

DOUBLE{}, SINGLE{} IEEE-Double/Single-Realformat lesen
DOUBLE{}=, SINGLE{}= ... schreiben
{ DO }= }, { SI }= }

Syntax (Lesen aus Adresse):
Realvar=DOUBLE{gerade Adresse} => 8-Byte-IEEE-Realzahl
Realvar=SINGLE{gerade Adresse} => 4-Byte-IEEE-Realzahl
Syntax: (Schreiben an Adresse):
DOUBLE{gerade Adresse}=Wert => 8-Byte-IEEE-Realzahl
SINGLE{gerade Adresse}=Wert => 4-Byte-IEEE-Realzahl

Beim Schreiben wird der angegebene 8- bzw. 4-Byte-Wert als Wert im IEEE-Format interpretiert und so an die angegebene Adresse geschrieben. Soll ein Wert gelesen werden, so wird intern der Inhalt der auf Adresse folgenden 4 bzw. 8 Byte als Wert im IEEE-Format interpretiert und in Realvar zurückgegeben.

Darüber, was das IEEE-Format ist, kann ich leider nur vage Vermutungen anstellen. Es gibt eine Epson-Schnittstelle, die den Namen "IEEE 488" trägt. Ich nehme daher an, daß dieses Datenformat speziell zum Informationsaustausch über eben diese Schnittstelle verwendet werden kann. Mir ist außerdem aufgefallen, daß eine glatte 2er-Potenz im Normalformat im IEEE-Double-Format immer 0 ergibt und bestimmte Werte (0.75, 1.5, 3, 6, 12, 24, 48, 96, 192) immer den IEEE-Wert 2.

Weitere Informationen zu diesem extravaganten Format kann ich Ihnen leider nicht liefern.

Beispiele:

```
For I=1 To 50
  A=Sqr(I)
  Print "IEEE 2^";I;" = ";Double{V:A}
Next I
'
For I=-50 To 50 Step 0.25
  A=I
  Print I,Double{V:A},
  Single{V:A*4}=A
```

```
    Print A
  Next I
```

Version 3.0

FLOAT{} 8 Byte in GFA-3.0-BASIC-Realformat lesen

FLOAT{}= {FL}= } 8 Byte in GFA-3.0-Realformat schreiben

Realvar=FLOAT{gerade Adresse} => Lesen (Funktion)
FLOAT{gerade Adresse}=Wert => Schreiben

Beim Schreiben wird der angegebene Wert als Wert im 8-Byte-Realformat des V3.0-GFA-BASICs interpretiert und so in die - auf Adresse folgenden - 8 Bytes geschrieben. Soll ein Wert gelesen werden, so wird intern der Inhalt der auf Adresse folgenden 8 Byte als V3.0-Realwert interpretiert und in Realvar zurückgegeben.

Version 3.0

INT{} /WORD{} 2 Byte als Vorzeichen-Integer lesen

INT{}= /WORD{}= ... schreiben

{WO}= }

Intvar=INT{Adresse} => Lesen (Funktion)
Intvar=WORD{Adresse} => Lesen (Funktion)
INT{Adresse}=Wert => Schreiben (Befehl)
WORD{Adresse}=Wert => Schreiben (Befehl)

Beim Schreiben wird der angegebene Wert als Wert im vorzeichenbehafteten 2-Byte-Integerformat interpretiert und so in die - auf Adresse folgenden - 2 Bytes geschrieben. Soll ein Wert gelesen werden, so wird intern der Inhalt der auf Adresse folgenden 2 Byte als vorzeichenbehafteter 2-Byte-Integerwert interpretiert und in der Aufnahmevariablen Intvar (beliebiger Typ) zurückgegeben.

Vorzeichenbehaftete 2-Byte-Werte können nur im Bereich von -32768 (&X1000000000000000) bis 32767 (&X0111111111111111) liegen. Ist also das höchste Bit (Bit 15) der beiden auf Adresse folgenden 16 Bits gesetzt, so ist der gelieferte Wert negativ. Der sich aus den untersten 15 Bits ergebende positive Normal-Integerwert (Wert And (2^15-1)) wird dann zu -32768 addiert und ergibt so den gelieferten Minuswert.

Wird als Intvar eine Byte-Variable (z.B. Var|) angegeben, so kann damit nur ein vorzeichenloser Wert von 0 - 255 aufgenommen werden.

Bei INT{} und WORD{} handelt es sich um zwei Namen derselben Funktion bzw. desselben Befehls. WORD{} hat den Vorteil, daß die Angabe als Befehl auf z.B. WO 23456}=12 verkürzt werden kann.

Für die V2.xx-Schreib-Simulation kann ohne weiteres DPOKE verwendet werden, da intern bei Minus-Pokes automatisch der richtige Komplementwert gebildet wird. Dabei dürfen jedoch keine Werte, die kleiner als -32768 oder größer als 32767 sind, verwendet werden, da sonst die Adreßinhalte "per Hand" komplementiert werden müßten, um beim Lesen korrekte Ergebnisse zu erhalten.

Als V2.xx-Simulation der Lese-Funktion kann eine relativ einfache DEFFN-Funktion eingesetzt werden:

```
A%=0
For I%=0 To 15                          ! 16 Bits
  Dpoke Varptr(A%)+2,-(2^I%)            ! löschen (-)
  Print Right$(String$(16,"0")+Bin$(A%),16),
  Print @Word(Varptr(A%)+2)             ! Word-Inhalt lesen
  ' in V3.0:  Print Word{V:A%+2}        ! (vorzeichenrichtig)
Next I%
Deffn Word(Adr%)=32768*((Dpeek(Adr%) And 2^15)<>0)+...
                          ... +(Dpeek(Adr%) And 32767)
```

PEEK, DPEEK, LPEEK

Speicherinhalt auslesen (Supervisor-Modus)

Var=PEEK(Adresse) => Liest ein Byte
Var=DPEEK(gerade Adresse) => Liest zwei Byte (Word)
Var=LPEEK(gerade Adresse) => Liest vier Byte (Long)

Ab Adresse werden dem Format entsprechend viele (1, 2 oder 4) Bytes gelesen. Bei DPEEK() und LPEEK() dürfen nur gerade Adressen verwendet werden. Die Funktion arbeitet generell im Supervisor-Modus. D.h. daß - im Gegensatz zu BYTE{}, CARD{} und LONG{} - auch Speicherstellen unterhalb der Adresse 2048 gelesen werden können.

Was wäre ein BASIC ohne PEEKs und POKEs?. Wie ein Rennwagen ohne Motor! Wer sich mit dem Programmieren in semiprofessionelle oder gar professionelle Höhen schwingen will, wird sich noch sehr intensiv mit den Möglichkeiten dieser Funktionen und Befehle auseinanderzusetzen haben. Was man damit alles machen kann, können Sie hier im Buch an (fast) unzähligen Stellen nachvollziehen. Unter anderem finden Sie Beispiele unter DATA in den Prozeduren Pcode und Scode, unter DEFMOUSE in der Prozedur Dmouse und in 9.5.1 "Organisation eines PUT-Strings".

POKE,DPOKE,LPOKE

{ PO,DP,LP } **Speicherinhalt ändern (User-Modus)**

POKE Adresse,Byte **=> Schreibt ein Byte**
DPOKE gerade Adresse,Word **=> Schreibt zwei Byte (Word)**
LPOKE gerade Adresse,Long **=> Schreibt vier Byte (Long)**

Schreibt im User-Modus den angegebenen Wert (Byte, Word, Long) im jeweiligen Format in die ab Adresse folgenden 1, 2 oder 4 Bytes. Bei DPOKE und LPOKE dürfen nur gerade Adressen verwendet werden.

Stellen Sie beim Experimentieren mit den POKEs sicher, daß Sie nicht unbeabsichtigt in wichtige Bereiche des Sytems oder des Interpreters hinein'poken'. Auch wenn das nicht immer sofort mit einem Absturz endet, können durch falsche Daten in diesen Bereichen Fehlfunktionen ausgelöst werden, die sich dann erst nach geraumer Zeit bemerkbar machen. Diese Fehlfunktionen sind im Endeffekt wesentlich gefährlicher als ein Absturz, da dadurch auch beim Laden und Speichern Daten unbemerkt "beschädigt" werden können, während man sich in Sicherheit wähnt.

Lassen Sie also bitte grundsätzlich bei allen Speichermanipulationen äußerste Vorsicht walten! Sollte Ihnen ein POKE mal "verloren" gehen, (wenn also seine Wirkung nicht nachvollziehbar ist), ist es in den meisten Fällen angebracht, seine Daten (Programm etc.) so schnell wie möglich auf Disk zu sichern und dann zum Resetknopf zu greifen, ehe Schlimmeres folgt.

Beispiele zu den POKEs finden Sie unter anderem unter DATA in Prozedur Pread, unter DEFTEXT, sowie unter PUT in der Prozedur Size.

SPOKE,SDPOKE,SLPOKE { SP,SD,SL } Supervisor-Poke

SPOKE Adresse,Byte **=> Schreibt ein Byte**
SDPOKE gerade Adresse,Word **=> Schreibt zwei Byte**
SLPOKE gerade Adresse,Long **=> Schreibt vier Byte**

Für diese Befehle gelten dieselben Ausführungen wie zu POKE/DPOKE/LPOKE. Der Unterschied ist, daß hiermit auch auf den Supervisor-Bereich (<2048) zugegriffen und dessen Inhalt verändert werden kann (siehe unter BYTE{}, CARD{}, LONG{} in der Prozedur Slow).

11.3 Speicherverwaltung

Version 3.0

INLINE { INL } BASIC-interne Speicherreservierung

INLINE Adresse%,Bytes

Reserviert innerhalb des Programm-Listings einen Speicherbereich von maximal 32700 Bytes. Dazu wird in Bytes die gewünschte Größe angegeben. Adresse% ist eine 4-Byte-Integer-Rückgabevariable (keine Feldvariable). Trifft das Programm auf eine INLINE-Zeile, so wird darin vom BASIC die aktuelle Startadresse des INLINE-Speichers geliefert. Sonst geschieht nichts. Ob etwas im Speicher ist, bzw. was Sie mit der Adresse und diesem Speicher anfangen, hängt davon ab, was Sie bei der Programmerstellung in diesen Speicher geladen haben.

Die Besonderheit dieses Befehls besteht nämlich darin, daß schon der Editor auf die Eingabe einer INLINE-Zeile reagiert. Überall dort, wo eine INLINE-Zeile geschrieben wurde und durch <Return> oder eine andere Editorfunktion verlassen wird, wird im Programmspeicher ein Bereich mit der angegebenen Größe reserviert. Wenn Sie nun das Programm mit der Editorfunktion Save bzw. SAVE oder PSAVE abspeichern, werden diese INLINE-Speicher als Programmbestandteil mitsamt Inhalt mit abgespeichert. Beim nächsten Laden durch Load bzw. LOAD werden die INLINE-Speicher ebenfalls mitgeladen und sind dann unverändert wieder verfügbar.

Mit der Editorfunktion Save,A bzw. dem Befehl LIST wird nur die Zeile, aber nicht der INLINE-Speicher gesichert. Beim Laden eines ASCII-Listings mit Merge wird zwar wieder der Speicher reserviert, aber er ist dann ohne Inhalt.

In einer INLINE-Befehlszeile ist kein !-Kommentar möglich, da an Stelle des Kommentars intern der INLINE-Speicher gelegt wird. Beim ersten Anlegen eines Speichers wird dieser mit Null-Bytes gefüllt. Wird eine bestehende INLINE-Zeile aus dem Programm gelöscht, wird auch automatisch der Speicher wieder an das BASIC zurückgegeben.

Wird dagegen eine bereits bestehende INLINE-Zeile nachträglich verändert (neuer Variablename oder neue Speichergröße), fragt der Interpreter zuerst nach, ob der alte INLINE-Speicher gelöscht werden soll. Wird diese Abfrage mit "OK" beantwortet, wird der alte Speicher gelöscht und einer neuer - den neuen Angaben entsprechender - Speicher eingerichtet.

Steht der Cursor auf einer fertigen (schon gecheckten) INLINE-Zeile, kann man die <Help>-Taste drücken. Das können Sie sonst natürlich auch, nur es hat dann keinen Zweck. Drücken Sie also die <Help>-Taste während der Cursor auf einer INLINE-Zeile steht, so erscheint eine Menüzeile.

Mit Mausklick auf Load (<L>) kann eine beliebige Datei in den reservierten Bereich geladen werden (z.B. Maschinen-Code, Bilddateien etc.). Mit Save (<S>) wird der INLINE-Speicher auf Diskette gesichert. Dump (<D>) ermöglicht die Ausgabe des INLINE-Inhalts in 2-Byte-Hexwerten (mit Offset-Angabe) auf dem Drucker und Clear (<C>) löscht (ohne Sicherheitsabfrage) den INLINE-Speicher.

Beim Speichern wird der INLINE-Datei - sofern keine andere angegeben wurde - die Extension .INL verpaßt. Beim Laden wird ebenfalls die Auswahl-Vorgabe .INL voreingestellt. Es kann jedoch jede beliebige Datei geladen werden. Die zu ladende Datei sollte entweder genauso groß wie der INLINE-Speicher oder größer sein. Dateien, die den INLINE-Speicher nicht ganz füllen, werden mit der Meldung "File-Ende erreicht" abgewiesen. Dateien mit einer Länge, die größer als der INLINE-Speicher ist, werden bei der INLINE-Länge abgeschnitten.

Noch einmal das Ganze in Stichpunkten:

1. INLINE-Zeile schreiben (Rückgabe-Variable und Größe angeben); wenn der INLINE-Speicher gefüllt, gesichert, gedruckt oder gelöscht werden soll:
2. Cursor auf INLINE-Zeile und <Help> drücken.
3. Menü-Funktion wählen.

Da der INLINE-Speicher innerhalb des Listings liegt, kann man seine Startadresse "natürlich" auch dazu verwenden, das Programm-Listing zu beeinflußen. Dazu müßte man sich allerdings mit den GFA-Opcodes genauer auskennen, womit ich leider nicht dienen kann. Aber mit dieser Starthilfe wird es sicher in Zukunft mehrere Insider geben, die sich hier noch manche Tricks und Kniffe einfallen lassen.

```
Inline Adr%,100
' Hier ist die Zeile hinter dem INLINE-Speicher
Print "INLINE-Adresse: ";Adr%
A$=Char{Adr%+101}
For I%=1 To Len(A$)
  Out 5,Asc(Mid$(A$,I%,1))
Next I%
```

Version 3.0

MALLOC() — System-Speicher-Reservierung

Back=MALLOC(Anz)

- Implementierung der GEMDOS-Funktion 72 -
Reserviert Anz Bytes oberhalb von HIMEM, bzw. oberhalb des zuletzt mit MALLOC reservierten (oder mit MSHRINK reduzierten) Speicherbereichs. Der Bereich ist anschließend gegen System-Zugriffe geschützt (alloziert/zugeteilt). Als Rückgabewert erhält man bei durchgeführter Reservierung die Startadresse des reservierten Bereichs (bei Fehler 0). Diese Adresse sollte man sich unbedingt merken, damit der Speicher auch wieder freigegeben bzw. reduziert werden kann.

Wird in Anz eine -1 übergeben, erhält man die Größe des noch verfügbaren Speichers. Vor Programmende sollte ggfs. MFREE() ausgeführt werden, da der reservierte Speicher auch durch RESERVE nicht mehr zurückgeholt werden kann.

Generell sind die Funktionen MALLOC(), MSHRINK() und MFREE() sehr kontrolliert einzusetzen. Kann ein allozierter Bereich nicht mehr korrekt freigegeben werden, ist damit der ganze Speicher ab der ersten, nicht freigegebenen MALLOC()-Adresse für das BASIC gesperrt.

Der durch MALLOC() vergebene Speicher ist auch gegen Zugriffe des GEM gesperrt. Damit das GEM jedoch genügend Platz für seine Arbeit hat, sollte unter dem Bildschirmspeicher immer ein Block von 16384 Bytes freigehalten werden (siehe auch unter RESERVE).

Als V2.xx-Funktion:

```
Back%=@Malloc(Anz)
Deffn Malloc(Size%)=Gemdos(72,L:Size%)
```

Diese DEFFN-Funktion ist mit MALLOC() absolut identisch.

Beispiel:

```
Sysfont(1)        ! 8*8-Font einschalten (siehe unter ASC()).
'                 ! (Prozedur Sysfont einbinden)
Print "Anfangszustand:"
Print "FRE(0)=";Fre(0);" / HIMEM=";Himem
Print "freier Speicher hinter HIMEM:";Malloc(-1)
Print
```

```
'                                              HIMEM->|16384|
' .------.-----.----.-------------------------.------|-----|
' |System|BASIC|Prog|<---        FRE(0)     --->|Felder| GEM |
' '------'-----'----'-------------------------'------'-----'
'                                                    XBIOS(2)-^
'
Print "Zustand nach 200 KByte-RESERVE:"
Reserve 200000            ! 200 Kilobyte für BASIC reservieren
Print "FRE(0)=";Fre(0);" / HIMEM=";Himem
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                            HIMEM->|<--     MALLOC(-1)     -->|
' .------.-----.----.------.------|------------------.-----|
' |System|BASIC|Prog|200KB |Felder|<--     Frei    -->| GEM |
' '------'-----'----|------|------'------------------'-----'
'                   |FRE(0)|                        XBIOS(2)-^
'
A1%=Malloc(20000)         ! 20 Kilobyte (Block 1) allozieren
Print "Startadresse des 1. MALLOC-Bereichs (20 KByte):";A1%
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                            HIMEM->| alloz.|<-  MALLOC(-1) ->|
' .------.-----.----.------.------|-------|-----------.-----|
' |System|BASIC|Prog|200KB |Felder|   M1  |<-  Frei ->| GEM |
' '------'-----'----|------|------'-------'-----------'-----'
'                   |FRE(0)|                        XBIOS(2)-^
'
A2%=Malloc(10000)         ! 10 Kilobyte (Block 2) allozieren
Print "Startadresse des 2. MALLOC-Bereichs (10 KByte):";A2%
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                            HIMEM->|<alloziert>|<MALLOC(-1) >|
' .------.-----.----.------.------|-------.---|-------.-----|
' |System|BASIC|Prog|200KB |Felder|   M1  | M2|<Frei >| GEM |
' '------'-----'----|------|------'-------'---'-------'-----'
'                   |FRE(0)|                        XBIOS(2)-^
'
~Mshrink(A1%,10000)       ! Block 1 auf 10 Kilobyte reduzieren
Print "1. MALLOC-Bereich reduziert (MSHRINK 10 KByte)"
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                            HIMEM->|<alloziert>|<MALLOC(-1) >|
' .------.-----.----.------.------|---.---.---|-------.-----|
' |System|BASIC|Prog|200KB |Felder| M1| * | M2|<Frei >| GEM |
' '------'-----'----|------|------'---'-|-'---'-------'-----'
'                   |FRE(0)| Frei/MFREE-'           XBIOS(2)-^
'
A3%=Malloc(10000)         ! 10 Kilobyte (Block 3) allozieren
Print "Startadresse des 3. MALLOC-Bereichs (10 KByte):";A3%
Print "(zwischen 1. und 2. MALLOC-Bereich !)"
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                            HIMEM->|<alloziert>|<MALLOC(-1) >|
' .------.-----.----.------.------|---.---.---|-------.-----|
' |System|BASIC|Prog|200KB |Felder| M1| M3| M2|<Frei >| GEM |
' '------'-----'----|------|------'---'---'---'-------'-----'
'                   |FRE(0)|                        XBIOS(2)-^
'
~Mshrink(A3%,2000)        ! Block 3 auf 2 Kilobyte reduzieren
```

```
Print "3. MALLOC-Bereich reduziert (MSHRINK 2 KByte)"
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                         HIMEM->|<alloziert>|<MALLOC(-1) >|
'  .------.-----.----.------.------|---.--.-.--|-------.-----|
'  |System|BASIC|Prog|200KB |Felder| M1|M3|*|M2|<Frei >| GEM |
'  '------'-----'----|------|------'---'--'|'--'-------'-----'
'                    |FRE(0)|  Frei/MFREE-'        XBIOS(2)-^
'
A4%=Malloc(2000)         ! 2 Kilobyte (Block 4) allozieren
Print "Startadresse des 4. MALLOC-Bereichs (2 KByte):";A4%
Print "(zwischen 3. und 2. MALLOC-Bereich !)"
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                         HIMEM->|<alloziert >|<MALLOC(-1)>|
'  .------.-----.----.------.------|---.--.--.--|-------.-----|
'  |System|BASIC|Prog|200KB |Felder| M1|M3|M4|M2|<Frei>| GEM |
'  '------'-----'----|------|------'---'--'--'--'-------'-----'
'                    |FRE(0)|                      XBIOS(2)-^
'
A5%=Malloc(20000)         ! 20 Kilobyte (Block 5) allozieren
Print "Startadresse des 5. MALLOC-Bereichs (20 KByte):";A5%
Print "(hinter 2. MALLOC-Bereich !)"
Print "noch nicht vergebener Speicher hinter HIMEM:";Malloc(-1)
Print
'                         HIMEM->|< alloziert  >|< MALLOC >|
'  .------.-----.----.------.------|--.--.--.--.--.----.-----|
'  |System|BASIC|Prog|200KB |Felder|M1|M3|M4|M2|M5|Frei| GEM |
'  '------'-----'----|------|------'--'--'--'--'--'----'-----'
'                    |FRE(0)|                      XBIOS(2)-^
'
~Mfree(A1%)       ! Block 1 freigeben
~Mfree(A2%)       ! Block 2 freigeben
~Mfree(A3%)       ! Block 3 freigeben
~Mfree(A4%)       ! Block 4 freigeben
~Mfree(A5%)       ! Block 5 freigeben
Reserve           ! BASIC-Speicher restaurieren
Print "BASIC-Speicher wieder restauriert. HIMEM=";Himem
```

Wenn man dieses Programm ablaufen läßt, könnte man den Eindruck bekommen, daß das alles recht unproblematisch läuft. Wenn man die Zuteilungen sehr kontrolliert einsetzt, ist es das auch. Es kann allerdings auch geschehen, daß - obwohl man weiß, daß noch ein großer Block frei sein müßte - MALLOC(-1) trotzdem wesentlich weniger Speicherplatz anbietet.

Der Fall tritt dann ein, wenn nach einem MSHRINK() ein MALLOC-Speicher angefordert wird, der größer als der durch MSHRINK() freigemachte Speicher ist. Dieser neue Block wird dann (wenn der Speicher hinter dem höchsten MALLOC-Block noch ausreicht) nicht auf die ggfs. durch MSHRINK() freigemachten Blöcke verteilt, sondern hinter den höchsten MALLOC-Block eingerichtet. Danach sind alle durch MSHRINK() freigemachten Blöcke **nicht** mehr zugänglich.

Im obigen Beispiel kann man diesen Effekt beobachten, wenn z.B. in der 3. MALLOC-Zuteilung nicht 10000 Bytes (genau die Größe zwischen M1 und M2), sondern 11000 Bytes angefordert würden. Der Speicher würde anschließend so aussehen:

```
                         HIMEM->|< alloziert  >|MALLOC(-1)|
.------.-----.----.------.------|---.---.---.--|----.-----|
|System|BASIC|Prog|200KB |Felder| M1| X | M2|M3|Frei| GEM |
'------'-----'----|------|------'---'-|-'---'--'----'-----'
                  |FRE(0)|   gesperrt-'          XBIOS(2)-^
```

Der hier mit X gekennzeichnete Bereich wäre nun bis zum nächsten Reset gesperrt und könnte auch nicht mehr durch MFREE() freigemacht werden. Würde man M1, M2 und M3 durch MFREE() freigeben und anschließend durch RESERVE (V3.0) bzw. RESERVE XBIOS(2)-16384-HIMEM+FRE(0) (V2.xx) versuchen, den BASIC-Speicher komplett zu restaurieren, erhielte man anschließend folgendes Bild:

```
                             HIMEM->|   |<-  MALLOC(-1) ->|
.------.-----.----.----------.------.---.-----------.-----|
|System|BASIC|Prog|< FRE(0) >|Felder| X |<-  Frei ->| GEM |
'------'-----'----|----------'------'-|-'-----------'-----'
                             gesperrt-'          XBIOS(2)-^
```

Weitere Beispiele hierzu finden Sie in den Beispielprogrammen zur AES-Library.

Version 3.0

MFREE() — MALLOC()-Speicher freigeben

Back=MFREE(Adresse)

- Implementierung der GEMDOS-Funktion 73 -

Gibt den durch MALLOC() reservierten Speicher wieder frei. In Adresse wird die Startadresse des freizugebenden Bereichs angegeben (bei MALLOC()-Aufruf merken). Wurde die Funktion ohne Fehler ausgeführt, erhält man als Rückgabe eine Null.

Andernfalls -40 (ungültige Speicherblockadresse).

Die Freigabe durch die MFREE() ist auch dann notwendig, wenn mit der Funktion Var=EXEC(3,...) ein Programm in den Speicher geladen wurde. Da dieses Programm nicht gestartet wird, kann es somit den belegten Speicher auch nicht wieder freigeben. Die bei EXEC(3...) in Var zurückgegebenen BASEPAGE-Adresse kann dann für MFREE()

verwendet werden. Bei Verwendung des Befehls EXEC 3,... ist diese Freigabe nicht möglich, da die BASEPAGE-Adresse anschließend nicht bekannt ist (siehe Beispiele zu EXEC).

Beispiele und weitere Informationen finden Sie unter MALLOC() sowie in den Beispielprogrammen zur AES-Library.

Als V2.xx-Funktion:

```
Back%=@Mfree(Adresse)
Deffn Mfree(Adr%)=Gemdos(73,L:Adr%)
```

Diese DEFFN-Funktion ist mit MFREE() absolut identisch.

Version 3.0

MSHRINK() — MALLOC()-Speicher einschränken

Back=MSHRINK(Adresse,Anz)

- Implementierung der GEMDOS-Funktion 74 -
Schränkt einen durch MALLOC() reservierten Speicherblock ein. In Adresse wird die Startadresse des zu vermindernden Speicherblocks angegeben (bei MALLOC()-Aufruf merken).

Anz enthält die gewünschte neue Größe des Blocks. Der freiwerdenden Bereich wird an das System zurückgegeben und kann ggfs. durch erneutes MALLOC() wieder zugeteilt werden. Wurde die Funktion ohne Fehler ausgeführt, erhält man als Rückgabe eine Null.

Andernfalls -40 (ungültige Speicherblockadresse) oder -67 (Speicherblockfehler, wenn Anz größer ist, als der zu reduzierende MALLOC()-Block). Beispiele und weitere Informationen finden Sie unter MALLOC() sowie in den Beispielprogrammen zur AES-Library.

Als V2.xx-Funktion:

```
Back%=@Mshrink(Adresse,Anzahl)
Deffn Mshrink(Adr%,Size%)=Gemdos(74,0,L:Adr%,L:Size%)
```

Diese DEFFN-Funktion ist mit MSHRINK() absolut identisch.

RESERVE { RESE } BASIC-Arbeitsspeicher festlegen

RESERVE Anz
RESERVE [[-]Anz] **(-> nur V3.0)**

V2.xx

Anz gibt die neu einzurichtende Größe des BASIC-Arbeitsspeichers in Bytes an. Diese Größe wird intern zum aktuell benötigten Arbeitsspeicher hinzuaddiert. Bei Speicher-Verminderung kann der oberhalb des BASIC-Speichers enstandene, unbelegte Speicher frei verwendet werden (für Resource-Files, Maschinen-Code etc.). Ursprungszustand wieder herstellen:

```
Reserve @Rmax       ! Gleiche Wirkung wie RESERVE
'                   ! ohne Parameter in V3.0
Deffn Rmax=Xbios(2)-Himem-16384+Fre(0)
```

V3.0

Ist Anz positiv, gilt hier die V2.xx-Erklärung analog. In V3.0 ist es auch möglich, einen Minuswert anzugeben. Mit diesem wird bestimmt, um welche Byte-Zahl der BASIC-Speicher am oberen Ende reduziert werden soll. Wird RESERVE ohne Parameter verwendet, wird (wenn möglich) der Ursprungszustand nach dem V2.xx-Schema (RESERVE @Rmax) automatisch wieder hergestellt.

Bei allen RESERVE-Aufrufen sind evtl. mögliche Bereichs-Kollisionen mit dem Gemdos-MALLOC bzw. - in V3.0 - mit der MALLOC-Funktion zu beachten.

Bei einer Verminderung des BASIC-Speichers kann der damit oberhalb des BASIC-Speichers enstandene freie Speicherbereich nach Belieben verwendet werden (siehe MALLOC()). Befehle wie BMOVE, BLOAD, BITBLT oder BGET#, bzw. RC_COPY bieten sich dazu an. Zum anderen kann dieser freie Speicher auch zur Unterbringung von RSC-Dateien verwendet werden (siehe Beispiele zur AES-Library). Eine eigene Speicherverwaltung birgt sehr viele Vorteile in sich, da eine selbst organisierte Datenverwaltung per BMOVE wesentlich schneller und vielseitiger ist, als es die reine BASIC-Verwaltung zuläßt. So kann man z.B. Strings mit mehr als 32767 Bytes oder auch eine komplette Dateiverwaltung mit eigener Datenstruktur dort einrichten. Hier im Buch finden Sie eine solche selbstverwaltete Dateiverwaltung im Kapitel 23 über die RAM-Kartei.

Wird der BASIC-Speicher vergrößert, ist dabei zu beachten, daß das GEM und GEMDOS zur Abwicklung verschiedener Prozesse (Alert-Box, Fileselect-Box, DTA etc.) auf einen Speicherbereich direkt unterhalb des Bildschirmspeichers (XBIOS(2)) zugreift.

Das BASIC setzt die obere Grenze des freien BASIC-Speichers bei Start automatisch 16384 Byte unterhalb des Video-RAMs. Es wäre also möglich, den Speicher bis zum Video-RAM auszuschöpfen (RESERVE FRE(0)+16384). In diesem Fall wären HIMEM und XBIOS(2) identisch. Wird so verfahren, hat das GEM keine Möglichkeit mehr z.B. den Hintergrund der FILESELECT-Box zwischenzuspeichern.

Der Speicher sollte deshalb in der maximalen Größe so gewählt werden, daß unterhalb von XBIOS(2) noch die erwähnten 16384 Bytes für das GEM und GEMDOS freigehalten werden.

Beispiele hierzu finden Sie unter anderem unter MALLOC(), EXEC, BMOVE und in der Prozedur Slow unter BYTE{}/CARD{}/LONG{}.

11.4 Zeigeroperationen

***** **Variablen-Pointer**

Var=*Var
Var=*Feld()

Wird ein Variablenname Var, bzw. Feld() beliebigen Typs (siehe TYPE) auf diese Art als Zeiger gekennzeichnet, wird nicht der Variableninhalt, sondern die Variablenadresse übergeben. Bei Feldern und Strings ist dies der zugehörige Descriptor (-> ARRPTR(X() oder ARRPTR X$), bei numerischen Variablen die Adresse, an welcher der Variableninhalt zu finden ist.

Beispiel 1:

```
A%=12          ! A% mit 12 belegen
B%=*A%         ! Pointer auf A% in B% speichern
*B%=33         ! Indirekt A% mit 33 belegen
Print "Variablenadresse A% (über Pointer): ";*A%
Print "Variablenadresse A% (über VARPTR) : ";Varptr(A%)
Print "Variablenadresse A% (in B%)       : ";B%
Print "Variableninhalt A%  (direkt)      : ";A%
Print "Variableninhalt A%  (indirekt)    : ";Lpeek(B%)
```

Beispiel 2:

```
Var$="GFA-BASIC"              ! String-Variable belegen
Var%=5                        ! 4-Byte-Integervariable belegen
Gosub Proc(*Var$,*Var%)       ! Proc-Aufruf mit Pointer-Vars
Print Var$,Var%               ! Globale Variablen ausgeben
Procedure Proc(Para1%,Para2%) ! Kopf mit Pointer-Aufnahme
  Local Lvar$,Lvar,Lvar%,I%   ! Lokale Variablen vorbereiten
  For I%=0 To Dpeek(Para1%+4)-1 ! String-Länge aus Descriptor
    Z%=Peek(Lpeek(Para1%)+I%) ! Zeichen lesen (mittels
    '                         ! des Descriptors durch PEEK)
    Lvar$=Lvar$+Chr$(Z%)      ! Lokalen String bilden
  Next I%                     ! Nächstes Zeichen
  If Type(Para2%)=0           ! Para2%=Realvariable? (siehe TYPE())
    Bmove Para2%,Varptr(Lvar),6 ! 6 Bytes in die lokale
    '                           ! Realvariable übertragen
    *Para1%=String$(3,Lvar$)+" / 2^"+Str$(Lvar)+" = "
    '                         ! String bilden und zurückgeben
    *Para2%=2^Lvar            ! Ergebnis berechnen und zurückgeben
  Endif
  If Type(Para2%)=2
    Lpoke Varptr(Lvar%),Lpeek(Para2%) ! 6 Bytes in die lokale
    '                         ! Integer-Variable übertragen
    *Para1%=String$(3,Lvar$)+" / 2^"+Str$(Lvar%)+" = "
    '                         ! String bilden und zurückgeben
    *Para2%=2^Lvar%           ! Ergebnis berechnen und zurückgeben
  Endif
Return
```

Das letzte Beispiel zeigt die einzige Möglichkeit in V2.xx, wie man eine globale Variable gleichzeitig zur Datenübergabe und Datenrückgabe verwenden kann. Dazu muß in jedem Fall der Umweg über die Variablenadressen bzw. den String-Descriptor gegangen werden. Die TYPE()-Abfrage ist jedoch nur dann nötig, wenn die Prozedur von mehreren Stellen aus aufgerufen wird und dabei die Typen der übergebenen Pointer variieren können. Da man jedoch in den meisten Fällen weiß, welche Variablentypen auftreten können, kann man die entsprechenden lokalen Variablen auch ohne TYPE()-Abfrage vorbereiten und füllen. In Version V3.0 ist bei Übergabe von Parameter-Variablen an Prozeduren der VAR-Befehl vorzuziehen.

Für die Übergabe von Feldern steht in V2.xx eine weitere Variante zur Verfügung. Man kann einen Feld-Pointer mit einem prozedur-internen Feld tauschen (siehe SWAP) und somit auch globale Felder in lokale Felder übertragen. Dabei werden die Descriptoren beider Felder miteinander vertauscht. Ist die Arbeit in der Prozedur erledigt, werden die Felder wieder zurückgetauscht.

Beispiel 3:

```
Dim A$(2)                    ! DIM String-Feld 1
A$(0)="String 1"             ! Elemente..
A$(1)="String 2"             ! ..belegen
A$(2)="String 3"             !
@Xyz(*A$())                  ! Aufruf mit Pointer-Übergabe
For I%=0 To 2                ! 3 Strings
  Print A$(I%)               ! Verändert ausgeben
Next I%
Procedure Xyz(Feldpointer%)
  Dim Dummyfeld$(1)          ! DIM Feld 2 (nur in V2.xx nötig !!!)
  Swap *Feldpointer%,Dummyfeld$() ! SWAP-Pointer und Dummy-Feld
  For I%=0 To 2              ! 3 Strings
    Clr X$                   ! Zeichenpuffer löschen
    For J%=Len(Dummyfeld$(I%)) Downto 1  ! Rückwärts alle Zeichen
         X$=X$+Mid$(Dummyfeld$(I%),J%,1) ! Neu zusammenfügen
    Next J%
    Dummyfeld$(I%)=X$        ! Puffer ins Feld übernehmen
  Next I%                    ! Nächster String
  Swap *Feldpointer%,Dummyfeld$() ! Felder zurück-"swappen"
  Erase Dummyfeld$()         ! ERASE Feld 2 (nur in V2.xx nötig !!!)
Return
```

Weitere Beispiele zu dieser Feldvertauschung finden Sie unter GET in der Prozedur Menue, unter PUT in der Prozedur Gplane und unter SSORT in der Prozedur Sort.

Sonstige Beispiele zur Verwendung von Pointern finden Sie unter DEFMOUSE (Dmouse), RIGHT$(), INSTR(), PSET, CALL, BMOVE, GOSUB (Reso) und unter DATA (Pread und Ersetzer).

Für die Variablen- und Feldübergabe in V3.0 eignet sich dagegen VAR. Hiermit ist es möglich, die Variablen und Felder "direkt" zu übergeben. Sie sind dann gleichzeitig Übergabe- und Rückgabevariablen. Eine Übernahme der Variableninhalte wie in Beispiel 2 ist hier nicht mehr nötig.

ARRPTR — String-/Feld-Descriptoradresse ermitteln

Var=ARRPTR(Var$)
Var=ARRPTR(Feld())

Liefert die Anfangsadresse des String-, bzw. Feld-Descriptors (siehe Erläuterungen in Kapitel 24.2 "Variablenorganisation/-typen")

VARPTR

Variablen-Adresse ermitteln

In V3.0: { V: }

Var=VARPTR(Var)

Liefert bei numerischen Variablen deren Adresse, bzw. bei String-Variablen die Adresse des ersten Zeichens Var steht für jede beliebige Variable (auch Feldelement). In Version V3.0 kann auch V: als Abkürzung verwendet werden (z.B. PRINT V:A$).

Beispiel:

```
A$="BASIC"          ! String setzen
Adr%=Varptr(A$)     ! String-Adresse holen
For I%=0 to 5       ! 5 Zeichen
  Print Chr$(Peek(A_dresse%+I%)); ausgeben
Next I%
```

12. Programmkontrolle

12.1 Programmstart und -ende

CONT { CON } Programm (nach STOP-Befehl) fortsetzen

CONT

Wurde der Programmlauf mit STOP unterbrochen, kann durch CONT im Direktmodus das Programm in der Zeile nach dem STOP-Befehl fortgesetzt werden. CONT ist nicht möglich, wenn nach STOP entweder CLEAR verwendet, das Programm-Listing verändert oder neue Variablen eingeführt wurden.

In manchen Fällen kann es notwendig sein, in ON BREAK GOSUB- oder ON ERROR GOSUB-Prozeduren CONT einzusetzen. Ob ein solcher Fall eintritt, hängt von verschiedenen Umständen ab, die sich nicht immer konkret vorhersagen lassen. Wenn es Ihnen passiert, daß das Programm in solchen Prozeduren "hängenbleibt", ist es gut, daß Sie von dieser Möglichkeit schon einmal gehört haben.

EDIT { ED } Programm beenden

EDIT

Hat dieselbe Wirkung wie END (siehe dort). EDIT kehrt jedoch ohne Vorwarnung direkt zum Editor (Interpreter), bzw. zum Desktop (Compilat) zurück.

END Programm beenden

END

Bewirkt Abbruch des aktuellen Programms. Variableninhalte/offene Dateien bleiben im Interpreter bis zur nächsten Programmänderung bzw. bis zum nächsten CLEAR erhalten/geöffnet und können im Direktmodus weiter angesprochen werden.

Das Programm kann danach nicht durch CONT fortgesetzt werden. Im Interpreterbetrieb erscheint eine Programm-Ende-Meldung, nach welcher zum Editor zurückgekehrt wird. Compilate kehren ohne

Ende-Meldung direkt zum Desktop bzw. zum aufrufenden Programm (siehe EXEC) zurück.

QUIT { Q } Programmende (Rückkehr zum Desktop)

QUIT
QUIT [x] **(nur V3.0)**

QUIT ist identisch mit SYSTEM und bewirkt, daß das Programm ohne jegliche Sicherheitsabfrage zum Desktop, bzw. (evtl. bei Compilaten) zum Aufrufer (EXEC) zurückkehrt.

In Version V3.0 kann in x ein 16-Bit-Wert angegeben werden, der an das aufrufende Programm (über D0) zurückgegeben und dann dort ausgewertet werden kann (siehe EXEC als Funktion).

Allgemeine Konvention:

x=0	Programm wurde korrekt - ohne Error -verlassen.
x>0	BASIC- oder Bombenfehler aufgetreten (ERR = 0 bis 109).
x<0	GEMDOS-, BIOS- oder XBIOS-Fehler aufgetreten (ERR= -1 bis -67).

Tritt ein TOS-, Bomben- oder BASIC-Fehler ein, könnte ggfs. QUIT ERR in einer Fehler-Abfangroutine (siehe ON ERROR GOSUB) als Programmende eingesetzt werden. So "weiß" ggfs. das aufrufende Programm, aus welchen Gründen das von ihm aufgerufene Programm beendet wurde.

RUN { RU } Programm starten

RUN
RUN "Programmname" **(nur V3.0)**

Startet das aktuelle Programm neu. Dabei werden sämtliche Variablen und der Bildschirm gelöscht. RUN kann auch im Direktmodus verwendet werden.

In Version V3.0 kann ein Programmname angegeben werden. Das angegebene Programm wird geladen und automatisch gestartet (vgl. CHAIN).

Manchmal ist es störend, daß beim Programmstart der Bildschirm gelöscht wird. Die beiden folgenden Prozeduren ermöglichen es, dieses CLS nach Belieben an- bzw. auszuschalten.

```
Procedure Clx(Flg%)                        ! Für V2.xx
  Local A$,Fnd%
  A$=Space$(10000)                         ! 10 KByte-Puffer
  Bmove Basepage+260,Varptr(A$),10000      ! 10 KByte in Puffer
  If Flg%=0                                ! CLS löschen?
    Fnd%=Instr(A$,Chr$(27)+"E")            ! Escape-Sequenz suchen
    If Fnd%>0                              ! Sequenz gefunden?
      Poke Basepage+260+Fnd%,Asc("H")      ! "H" schreiben
    Endif
  Else                                     ! CLS anschalten?
    Fnd%=Instr(A$,Chr$(27)+"H")            ! Escape-Sequenz suchen
    If Fnd%>0                              ! Sequenz gefunden?
      Poke Basepage+260+Fnd%,Asc("E")      ! "E" schreiben
    Endif
  Endif
Return
```

Da es mehrere V2.xx-Versionen gibt, kann die ausschlaggebende Escape-Sequenz an verschiedenen Stellen liegen. Deshalb wird sie in der ersten Prozedur zuerst gesucht und dann an der gefundenen Position verändert.

In V3.0 liegt die Sequenz 7443 Bytes hinter der Basepage. Dadurch vereinfacht sich das Verfahren in V3.0 natürlich.

```
Procedure Clx(Flg%)                        ! Für V3.0
  If Flg%=0                                ! CLS löschen?
    Poke Basepage+7443,Asc("H")
  Else                                     ! CLS anschalten?
    Poke Basepage+7443,Asc("E")
  Endif
Return
```

Wer grundsätzlich das Anfangs-CLS unterdrücken will, kann auch direkt den Interpreter, den Run-Only-Interpreter bzw. ein GFA-Compilat auf Diskette "patchen".

```
Open "U",#1,"GFABASIC.PRG"          ! Für Interpreter
'   Open "U",#1,"GFABASRO.PRG"      ! Für Run-Only-Interpreter
'   Open "U",#1,"BELIEBIG.PRG"      ! Für GFA-Compilate
'                                   ! (Programmnamen angeben)
Seek #1,31                          ! File-Pointer auf Byte 31
Out #1,Asc("H")                     ! Anfangs-CLS aus
'   Out #1,Asc("E")                 ! Anfangs-CLS an
Close #1                            ! Datei schließen
```

Wer seine Programme mit dem Run-Only-Interpreter betreiben (lassen) will, kann ein Programm ohne FILESELECT-Auswahl direkt durch das GFABASRO.PRG starten lassen. Dazu wird ab Byte 32 des Run-Only-Interpreters ein max. 63 Zeichen langer Programmname (ggfs. inkl. Pfad) abgelegt. Sobald der Interpreter gestartet wird, lädt er das angegebene Programm-File und startet es selbsttätig.

```
Open "U",#1,"GFABASRO.PRG"
Seek #1,32                        ! File-Pointer auf Byte 32
Print #1,"Progname.BAS";Chr$(0);  ! Für V2.xx ---. Name
' Print #1,"Progname.GFA";Chr$(0); ! Für V3.0  ---' eintragen
Close #1                          ! Datei schließen
```

Achten Sie darauf, daß der Programmname mit einem Null-Byte (C-Konvention) abgeschlossen wird. Wird nur ein Null-Byte übergeben, wird der normale Auswahlmodus wieder initialisiert.

STOP { ST } — Programm unterbrechen

STOP

Mit STOP kann der Programmlauf an jeder Stelle unterbrochen werden. Es erscheint eine ALERT-Box, durch die man das Programm fortsetzen kann oder in den Direktmodus gelangt.

Da keine Variablen gelöscht und keine Dateien geschlossen werden, kann aus dem Direktmodus heraus durch Eingabe einzelner Befehlszeilen schrittweise gearbeitet werden und anschließend das Programm ggfs. durch Eingabe von CONT im Direktmodus fortgesetzt werden (siehe CONT). Es kann auch zum Editor gewechselt werden. Solange dort keine Zeilen verändert werden, kann auch dann nach Rückkehr zur Direkt-Eingabeebene durch CONT das Programm wieder aufgenommen werden.

Beispiel:

```
Do                    ! Endlos-Schleife
  Inc Aa              ! Irgendeine Zählvariable +1
  If Aa>100           ! Variable größer 100?
    Clr Aa            ! Variable löschen
    Stop              ! Programmstop
  Endif
  Print Aa''          ! Wert ausgeben
Loop
```

SYSTEM { SYS } — Programmende (Interpreter verlassen)

SYSTEM
SYSTEM [x] **(nur V3.0)**

Ist identisch mit QUIT (siehe Erläuterungen dort).

12.2 Löschfunktionen

CLEAR {CLE} — Felder und Variablen löschen

CLEAR

Alle numerischen Variablen erhalten den Wert 0, alle String-Variablen werden zu Leer-Strings. Felder werden gelöscht und ihre Dimensionierung aufgehoben. CLEAR darf nicht in Prozeduren oder FOR/NEXT-Schleifen verwendet werden. Bei Programmstart wird CLEAR automatisch ausgeführt.

CLR — Einzelvariablen löschen

CLR Var [,Var%,Var$,...]

Es kann eine Liste von Variablen (keine Feldvariablen) übergeben werden, deren Inhalte gelöscht werden sollen, z.B.:

```
Clr A$,B%,C,D!
```

entspricht:

```
A$=""     B%=0     C=0     D!=0
```

CLS — Bildschirm löschen

CLS [#Kanal]

Löscht den Ausgabebildschirm, bzw. das jeweils geöffnete GEM-Fenster und setzt den Cursor auf Home (linke obere Ecke). Im Falle, daß durch die Option Kanal CLS in eine Diskettendatei geschrieben wird, wird beim Lesen dieser Datei der Bildschirm gelöscht, sobald der Lesezeiger auf CLS trifft.

NEW — Programmspeicher löschen

NEW

Löscht den BASIC-Arbeitsspeicher mitsamt dem Programm und seinen Variablen. Der Speicher ist für neue Anwendungen frei.

12.3 Zeitoperationen

DATE$ — Systemdatum ermitteln

Var$=DATE$
DATE$="Datum-String" **(nur V3.0)**

DATE$ ist eine reservierte String-Variable, die dieses aktuelle Systemdatum als Text-String im Format DD.MM.YYYY (D = Tag/M = Monat/Y = Jahr) enthält. In V3.0 ist es möglich, DATE$ ein neues Datum in "Datum-String" zuzuweisen. Das Format dieses Strings ist unter SETTIME beschrieben.

Wie Sie sicher wissen, kann im Kontrollfeld des Desktops das aktuelle Datum eingegeben werden. Diese Angabe wird systemintern ständig auf den aktuellen Stand gebracht. Haben Sie selber keine Veränderungen an dieser Einstellungen vorgenommen (siehe SETTIME), so erhalten Sie immer das jeweilige TOS-Versionsdatum.

Ein Beispiel zu DATE$ finden Sie ebenfalls unter SETTIME.

Version 3.0

DELAY { DELA } — 1/1Sek.-Wartefunktion

DELAY Sekunden

Sekunden bestimmt, wieviel Sekunden das Programm pausieren soll (sonst siehe PAUSE).

PAUSE { PA } — 1/50Sek.-Wartefunktion

PAUSE Dauer

Dauer bestimmt in 50stel Sekunden, wie lange das Programm pausieren soll. In dieser Zeit ist ausschließlich die Break-Funktion aktiv.

Andere Aktivitäten (ON ERROR GOSUB, ON MENU xxxx GOSUB, EVERY/AFTER GOSUB etc.) werden für die angegebene Dauer eingestellt.

Das Problem der 16-Bit-Computer (der 32-Bit-Computer erst recht) ist nicht mehr, daß sie so langsam sind und dadurch zwangsläufig Pausen schaffen, sondern daß sie so schnell sind, daß man ihnen

manchmal eine kleine Verschnaufpause aufzwingen muß, um bestimmte Programmläufe überhaupt noch kontrollieren zu können. In einigen Programmen finden Sie den Begriff "kleine Klickpause". Er kennzeichnet, was gemeint ist.

Stellen Sie sich vor, Sie lassen eine Schleife mit der Abbruchbedingung EXIT IF MOUSEK=1 enden und ein daran anschließender Block wird nur betreten, IF MOUSEK=1 ist. Selbst wenn dazwischen noch weitere Zeilen liegen, ist der Computer so schnell, daß der Mausklick zum Verlassen der Schleife gleichzeitig als Bedingungs-Erfüllung für den Eintritt in den IF MOUSEK=1-Block gewertet wird. Damit der Benutzer des Programms in solchen Fällen Zeit hat, die Maustaste wieder loszulassen, legt man zwischen die beiden Bedingungen eine kleine Pause (z.B. PAUSE 5), die dann im Pogrammverlauf von einem "Unwissenden" gar nicht bemerkt wird.

SETTIME {SETT} Uhrzeit und Datum einstellen

SETTIME Zeit$,Datum$

In Zeit$ und Datum$ wird die neue Systemzeit und das neue Systemdatum bestimmt. Es müssen beide Strings übergeben werden.

Europa-Format :

```
Zeit$  = "hh:mm:ss"    oder  "hhmmss"
Datum$ = "dd.mm.yyyy"  oder  "dd.mm.yy"
```

USA-Format (nur in V3.0 - siehe MODE):

```
Zeit$  = "hh:mm:ss"    oder  "hhmmss"    (wie oben)
Datum$ = "mm/dd/yyyy"  oder  "mm/dd/yy"
```

Die Jahresangabe kann auch zweistellig erfolgen (z.B. 86 für 1986), falls es sich um eine Angabe zwischen 1980 und 2079 handelt. Bei der Zeitangabe können die Sekunden (ggfs. inkl. Doppelpunkt) weggelassen werden. Die Sekunden werden dann auf Null gesetzt, z.B.:

```
SETTIME "15:37",               => Verändert nur die Uhrzeit
SETTIME "15:37:22","15.07.88"  => Verändert beide Einträge
SETTIME "","15.07.1988"        => Verändert nur das Datum
```

Wird das Format nicht korrekt eingehalten, werden die Angaben ignoriert und der alte Inhalt unverändert beibehalten. Die Sekunden der Zeitangabe werden übrigens nur in Zweierschritten (0, 2, 4, 6 etc.)

übernommen, ungerade Angaben werden auf die nächsthöhere gerade Zahl "gerundet".

Beispiel:

Im Lauf des Buches habe ich mir angewöhnt, "mundgerechte" Beispiele zu liefern. In diesem Sinne: eine Uhrzeit-/Datum-Aus- und Eingabe-Routine.

Hierzu ist nicht viel zu erklären. Es wird nur eine Cursor-Position übergeben, an welcher der/die Strings ausgegeben werden sollen. Ein Flag bestimmt darüber, ob nur der Uhrzeit-String oder zusätzlich auch der Datum-String angezeigt werden soll. Beide Strings können mit der Maus angeklickt werden und erwarten dann die Eingabe eines neuen Datum- bzw. Uhrzeit-Strings. Der Cursor kann durch die beiden Horizontal-Pfeiltasten (<-/->) gesteuert werden. Nach Eingabe eines neuen Strings wird die Eingabe durch <Return> oder <Esc> verlassen. Durch <Esc> wird die neue Eingabe ignoriert und der alte String wieder restauriert.Der/die Strings werden nur einmal bei Aufruf dargestellt. Die Routine kehrt also sofort wieder zum Programm zurück. Die Routine bleibt nur dann in der Eingabeschleife "hängen", wenn zum Zeitpunkt der Anzeige der Zeit- bzw. der Datum-String angeklickt wurde. Der Hintergrund wird **nicht** automatisch restauriert, dies ist also Ihre Sache.

Da die Eingabe in V2.xx sich als sehr schwierig erwies (drei getrennte Eingabebereiche zu je 2 Zeichen mit Vorgabe und Overwrite), mußte ich eine eigene Eingaberoutine schreiben, die sich auch zu anderen Zwecken weiterentwickeln läßt. So erhalten Sie unter SETTIME noch eine kleine Einführung in die Cursor-Steuerung und Einzelzeichen-Ein/Ausgabe.

```
Box 66,138,141,181      !-----. Kleinen Rahmen
Box 69,141,138,178            ! zeichnen (sieht
Box 70,142,137,177      !-----' ein bißchen netter aus!)
Do                      ! Endlos-Schleife
  @Timdat(10,10,1)      ! Aufruf (mit Änderungsmöglichkeit)
Loop
Procedure Timdat(X%,Y%,Flg%)
  ' Für Hires/Midres/Lowres und alle Versionen ab V2.0
  ' X%   = Cursor-Spalte, in der die Strings beginnen sollen
  ' Y%   = Cursor-Zeile der Anzeige. Der Datum-String wird
  '        ggfs. immer direkt unter der Zeit ausgegeben.
  ' Flg% = Flag
  '          0   = Nur Zeit-String anzeigen
  '          1   = Auch Datum-String anzeigen
  '          2   = Wie 0, jedoch ohne Änderungsmöglichkeit
  '          3   = Wie 1, jedoch ohne Änderungsmöglichkeit
```

```
  Local Yt%,J%             ! Lokale Variablen
  Yt%=Min(2,3-Xbios(4))    ! Y-Auflösungsteiler
  Print Chr$(27);"j";At(X%,Y%);Chr$(27);"p";
  ' Cursor-Position speichern, Cursor positionieren und
  ' Revers-Text einschalten
  For J%=1 To 8            ! 8 TIME$-Zeichen
    Out 5,Asc(Mid$(Time$,J%,1))+32*(Mid$(Time$,J%,1)<>":")
    ' Im VID:-Modus ausgeben. Die Zahlen werden dabei in den
    ' Digital-ASCII konvertiert.
  Next J%
  If Flg%=1 or Flg%=3      ! Datum auch anzeigen?
    Print At(X%,Y%+1);     ! Cursor positionieren
    For J%=1 To 10         ! 10 DATE$-Zeichen
      If J%<7 Or J%>8      ! Die 19 der Jahreszahl auslassen
        Out 5,Asc(Mid$(Date$,J%,1))+32*(Mid$(Date$,J%,1)<>".")
      Endif                ! Ausgeben
    Next J%
  Endif
  Print Chr$(27);"q";      ! Revers-Text aus
  If Flg%<2                ! Änderung zugelassen
    If Mousek              ! Maustaste gedrückt?
      If Mousex>X%*8-8 And Mousex<(X%+8)*8-8
        '                  ! X-Maus auf TIME-String?
        If Mousey>(Y%*16/Yt%-16/Yt%) And Mousey<Y%*(16/Yt%)
          '                ! Y-Maus auf TIME-String?
          @Eingabe(X%,Y%,Time$,*Zeit$) ! Dann zur Zeit-Eingabe
          Settime Zeit$,"" ! Neue Zeit setzen
        Else               ! Y-Maus nicht auf TIME-String!
          If Flg%=1 or Flg%=3 ! DATE-String ist angezeigt?
            If Mousey>(Y%*16/Yt%+1) And Mousey<Y%*(16/Yt%)+16/Yt%
              '                ! Y-Maus auf DATE-String?
            @Eingabe(X%,Y%+1,Date$,*Datum$) ! Zur Datum-Eingabe
              Settime "",Datum$ ! Neues Datum setzen
            Endif
          Endif
        Endif
      Endif
    Endif
  Endif
  Print Chr$(27);"k";      ! Cursor auf gespeicherte Position
Return
Procedure Eingabe(Xp%,Yp%,Dz$,Ptr%) ! Unterprozedur zu Timdat
  Local I%,Key%,Cnt%,Dz2$
  Dim Cp%(8)               ! ASCII-Wert-Puffer für Einzelzeichen
  Dz2$=Dz$                 ! Restore-String puffern
  Dz$=Left$(Dz$,6)+Right$(Dz$,2) ! Ggfs. bei DATE$ die
  '                        ! 19 aus der Jahreszahl auslassen
  Print At(Xp%,Yp%);       ! Cursor positionieren
  For I%=1 To 8            ! 8 Zeichen
    Cp%(I%-1)=Asc(Mid$(Dz$,I%,1)) ! ASCIIs puffern
    If I%<>3 And I%<>6     ! Ziffernposition?
      Poke Varptr(Dz$)+I%-1,(Cp%(I%-1)-32) ! Dann ASCII
      '                    ! auf Digital-Zeichen reduzieren
      Out 5,Cp%(I%-1)-32   ! und ausgeben
    Else                   ! Trennzeichenposition (: oder .) !
      Out 5,Cp%(I%-1)      ! Trennzeichen ausgeben
    Endif
  Next I%                  ! Nächste String-Position
  Print At(Xp%,Yp%);Chr$(27);"e"; ! Cursor neu positionieren
```

```
  '                       ! und anschalten
  Repeat                  ! Eingabe-Schleife
    Key%=Inp(2)           ! Auf <Taste> warten
    If Key%=>Asc("0") And Key%<=Asc("9") ! Zifferntaste?
      Out 5,Key%-32       ! Digital-Ziffer ausgeben
      Cp%(Cnt%)=Key%      ! ASCII-Puffer aktualisieren
      Inc Cnt%            ! Positionszähler +1
      If Cnt%=8           ! Auf letzter Position?
        Dec Cnt%          ! Zähler wieder -1
        Print Chr$(27);"D"; ! Cursor ein Zeichen zurück
      Endif
      If Cnt%=2 Or Cnt%=5   ! Cursor auf Trennzeichen?
        Inc Cnt%            ! Zähler nochmal +1
        Print Chr$(27);"C"; ! Cursor ein Zeichen weiter
      Endif
    Endif
    If Key%=203 And Cnt%>0  ! Pfeiltaste links gedrückt
      '                     ! und Cursor noch im String?
      Print Chr$(27);"D";   ! Cursor ein Zeichen zurück
      Dec Cnt%              ! Positionszähler +1
      If Cnt%=2 Or Cnt%=5   ! Cursor auf Trennzeichen?
        Print Chr$(27);"D"; ! Cursor noch ein Zeichen zurück
        Dec Cnt%            ! Positionszähler nochmal -1
      Endif
    Endif
    If Key%=205 And Cnt%>8  ! Pfeiltaste rechts gedrückt
      '                     ! und Cursor noch im String?
      If Cnt%<7             ! Cursor nicht auf letzter Position?
        Print Chr$(27);"C"; ! Cursor ein Zeichen vorwärts
        Inc Cnt%            ! Positionszähler +1
        If Cnt%=2 Or Cnt%=5 ! Cursor auf Trennzeichen?
          Print Chr$(27);"C"; !Cursor noch ein Zeichen weiter
          Inc Cnt%          ! Positionszähler nochmal +1
        Endif
      Endif
    Endif
  Until Key%=13 Or Key%=27  ! Exit, wenn <Return> oder <Esc>
  If Key%=13                ! <Return> gedrückt?
    Clr Dz$                 ! Alten String löschen
    For I%=0 To 7           ! 8 Zeichen
      Dz$=Dz$+Chr$(Cp%(I%)) ! Neu zusammenbinden
    Next I%                 ! Nächstes Zeichen
    *Ptr%=Dz$               ! Neuen String zurückgeben
  Else                      ! <Esc> gedrückt!
    *Ptr%=Dz2$              ! Restore-String zurückgeben
  Endif
  Erase Cp%()               ! ASCII-Pufferfeld löschen
  Print Chr$(27);"f";       ! Cursor wieder ausschalten
Return
```

TIMES$ System-Uhrzeit ermitteln

Var$=TIME$
TIME$="Zeit-String" **(nur V3.0)**

TIME$ ist eine reservierte String-Variable, die die aktuelle Uhrzeit als Text-String im Format hh:mm:ss (h = Stunde/m = Minute/s = Sekunde) enthält. In V3.0 ist es möglich, TIME$ eine neue Uhrzeit in Zeit-String zuzuweisen. Das Format dieses Strings ist unter SETTIME beschrieben. Die Sekunden der Zeitangabe werden übrigens in Zweierschritten erhöht.

Wie Sie sicher wissen, kann im Kontrollfeld des Desktops die aktuelle Tageszeit eingegeben werden. Diese Angabe wird systemintern ständig auf den aktuellen Stand gebracht. Haben Sie selber keine Veränderungen an dieser Einstellungen vorgenommen (siehe SETTIME), so gilt der Systemstart als 00:00:00 und Sie erhalten durch TIME$ die seit Systemstart vertrichene Zeit.

Im GFA-Editor der V3.0 ist dies die Zeitangabe, die Sie rechts oben auf dem Bildschirm sehen. Für Dauer-Computerer (wie mich) wäre es übrigens sicher angebracht gewesen, zusätzlich noch eine Weck-Zeit-Eingabe zu installieren, damit man nicht das Schlafengehen "verschläft". Bei mir müßte dann allerdings nach Erreichen der Weckzeit eine "Absturz-Zeitbombe" eingebaut werden, da ich bis jetzt jedes gesetzte Zeitlimit um Längen geschlagen habe.

Ein Beispiel zu TIME$ finden Sie ebenfalls unter SETTIME.

TIMER Laufzeit ermitteln

Var=TIMER

Reservierte Variable. Enthält die seit Systemstart verstrichene Zeit in 200stel Sekunden.

Der ST verfügt über einen Zeit-Zähler, der alle 1/200stel Sekunden Sekunden um 1 erhöht wird. Dieser Zähler beginnt zum Zeitpunkt des Systemstarts bei Null und erhöht sich also in jeder Sekunde um den Wert 200. Dabei ist es unerheblich, ob zwischenzeitlich irgendwelche Anwendungen ausgeführt werden. Der Zähler orientiert sich an dem konstant bleibenden Takt des Prozessors. D.h. also, daß Sie anhand dieses Zählers exakt feststellen können, wieviel Zeit seit dem Systemstart vergangen ist. Das heißt, wenn er nicht zwischenzeitlich

durch SLPOKE &H4BA,Timerwert verändert wurde. Außerdem läßt sich dieser TIMER hervorragend dazu verwenden, von einer bestimmten Zeitdauer abhängige Arbeiten ausführen zu lassen oder die Zeitdauer bestimmter Prozesse zu messen (Benchmark-Tests).

Beispiel 1 (Zeitanzeige):

```
Time=Timer              ! Timer puffern
Do                      ! Endlos-Schleife
  X%=(Timer-Time)/200   ! Differenz in Sekunden
  Print At(10,10);Right$(String$(3,"0")+Str$(X%),3);" Sek."
Loop
```

Beispiel 2 (Zeitmessung):

```
Time=Timer
Print "20000er Integer-FOR..NEXT-Leerschleife: ";
For I%=0 To 20000
Next I%
Print (Timer-Time)/200;" Sek."
Time=Timer
Print "20000er Real-FOR..NEXT-Leerschleife: ";
For I=0 To 20000
Next I
Print (Timer-Time)/200;" Sek."
Time=Timer
Print "20000er Integer-REPEAT..UNTIL-Leerschleife: ";
Clr I%
Repeat
  Inc I%
Until I%=20000
Print (Timer-Time)/200;" Sek."
Time=Timer
Print "20000er Integer-DO..LOOP-Leerschleife: ";
Clr I%
Do
  Inc I%
  Exit if I%=20000
Loop
Print (Timer-Time)/200;" Sek."
```

Beispiel 3 (Quasi-Multitasking):

```
Print "Bitte <Tasten> drücken"
Graphmode 3                  ! XOR-Modus für Auf/Zu-Box
Do
  Key$=Inkey$                ! Tatstatur abfragen
  If Key$>""                 ! Taste gedrückt?
    Print Key$;              ! Zeichen ausgeben
  Endif
  If Timer Mod 100=0         ! Jede 1/2 Sekunde
    For I%=10 To 100 Step 6  ! ----------.
      Box 110-I%,110-I%,120+I%,120+I%    |  Parallel-
    Next I%                              |-  Prozeß
    For I%=100 To 10 Step -6             |  laufen lassen
```

```
        Box 110-I%,110-I%,120+I%,120+I%     |
      Next I%                       |-----------'
    Endif
  Loop
```

12.4 Fehlerbehandlung

ERR — Fehler-Code ermitteln

Var=ERR

ERR ist eine reservierte Variable, die nach Auftreten eines Fehlers seine Identifikationsnummer enthält. Siehe 24.12 "GFA-BASIC-Fehlerliste".

Ein Beispiel finden Sie unter FATAL.

Version 3.0

ERR$ — Fehlertext liefern

Var$=ERR$(Index)

Die Funktion ERR$ liefert in V3.0 den Text der Fehlermeldung, deren Fehlerindex angegeben wurde (siehe 24.12 "GFA-BASIC-Fehlerliste"). Dieser Text ist im Format [Icon][Boxtext][Buttontext] und kann direkt an FORM_ALERT() übergeben werden.

Ein Beispiel finden Sie unter FATAL.

ERROR { ERR } — Fehler simulieren

In V3.0: { ER }

ERROR Fehlernummer

Fehlernummer steht für die Identifikationsnummer des zu simulierenden Fehlers (siehe 24.12 "GFA-BASIC-Fehlerliste").

Es wird entweder die entsprechende Fehlermeldung ausgegeben und das Programm beendet, oder es wird - wenn ON ERROR GOSUB aktiv ist - zu der dort angegebenen Prozedur verzweigt.

FATAL — Fehlerart ermitteln

Var=FATAL

Reservierte Variable. Es wird eine Unterscheidung zwischen "Normal"- und "Bomben"-Fehlern getroffen. Ist ein "Bomben-Fehler" aufgetreten (Adresse des zuletzt ausgeführten BASIC-Befehls ist nicht mehr bekannt) enthält sie eine -1. Bei allen anderen Fehlern enthält sie eine 0.

0 Allgemeiner BASIC-Fehler. Die Variablen- und Label-Adressen, sowie der GOSUB-Stapel sind intakt.

-1 Fataler Systemfehler. Die Adressenlage ist zerstört, was normalerweise eine bombigen Absturz zur Folge hat. In GFA-BASIC werden diese Fehler ebenfalls abgefangen, da das System bei den "normalen" Bomben-Errors die Registerinhalte in einen besonderen Bereich rettet, der - falls kein Total-Absturz (80 Bomben) eingetreten ist - zur Re-Initialisierung wieder ausgelesen werden kann. Dieser Rettungspuffer hat folgenden Aufbau:

Ab Adresse:	
$0380 (896)	Sind die geretteten Daten gültig, steht hier Longword 305419896.
$0384 (900)	Die alten Datenregister (D0 - D7) (8 Longwords).
$03A4 (932)	Die alten Adreßregister (A0 - A6) (7 Longwords).
$03C0 (960)	Alter Super-Stack-Pointer (1 Longword).
$03C4 (964)	Error-Index (Bomben-Anzahl) (1 Byte).
$03C8 (968)	Altes Register A7 (User-Stack-Pointer) (1 Longword).
$03CC (972)	16 alte Super-Stack-Words (? Longwords/? Words).

Für Interressierte läßt sich dieses Wissen natürlich prächtig dazu mißbrauchen, dem BASIC ab und zu über die Schulter zu schauen.

Allerdings habe ich den Eindruck, daß man sich auf FATAL nicht unbedingt verlassen kann. So führt ein DPOKE 11111,1 (ungerade DPOKE-Adresse) im Direktmodus zu FATAL = -1. Im Programm selbst konnte ich dagegen nur über den Error-Index ERR feststellen, ob eine Bombenfehler aufgetreten ist. FATAL lieferte in der Abfang-Routine grundsätzlich den Wert 0. Daß trotzdem ein fataler Error aufgetreten ist, läßt sich daran erkennen, daß das System die Register gerettet hat.

Beispiel:

```
On error gosub Fehler  ! Abfangroutine angeben
Poke 0,1               ! 2 Bomben: POKE im Supervisor-Bereich >--.
Label1:                ! Hier weitermachen                         |
Dpoke 11111,1          ! 3 Bomben: Ungerade DPOKE-Adresse >-----.|
```

```
Label2:                 ! Hier weitermachen                         ||
Monitor                 ! 4 Bomben: Monitor ohne Masch.-Progr.>-.  ||
Label3:                 ! Hier weitermachen                      | ||
Print Sqr(-100)         ! Error: Minus-Wurzel >-----------.      | ||
Print 0/0               ! Error: Division durch Null >---. |     | ||
Print "Programm geht ab hier weiter!"                    | |     | ||
Procedure Fehler        ! <-----------------<-----------'-'-----'''
  Print "ERR : ";Err'' ! ERR ausgeben
  ' ~Form_alert(1,Err$)! Für V3.0
  '    In V3.0 kann durch ERR$ der Original-Fehlertext
  '    an FORM_ALERT() übergeben, oder anderweitig verwendet
  '    werden.
  If Err>101            ! Bomben-Fehler?
    Print "Fatal-Error"
    Print "Inhalt von D0 bis D7 :"
    For I%=0 To 7       ! 8 Datenregister...
      Print Left$(Str$(Lpeek(900+I%*4))+String$(8,32),8);
    Next I%             ! ...ausgeben
    Print Chr$(10);Chr$(13);String$(60,"-")
    Print "Inhalt von A0 bis A6 :"
    For I%=0 To 6       ! 7 Adreßregister...
      Print Left$(Str$(Lpeek(932+I%*4))+String$(8,32),8);
    Next I%             ! ...ausgeben
    Print Chr$(10);Chr$(13);String$(60,"=")
    On error gosub Fehler ! Abfangroutine neu angeben      .
    If Err=102          ! 2-Bomben-Fehler?                 |
      Resume Label1     ! Dann zum Label1 ----------------'
    Endif                                                    .
    If Err=103          ! 3-Bomben-Fehler?                   |
      Resume Label2     ! Dann zum Label2 ------------------'
    Endif                                                      .
    If Err=104          ! 4-Bomben-Fehler?                     |
      Resume Label3     ! Dann zum Label3 --------------------'
    Endif
  Else                  ! Normaler Error!
    On error gosub Fehler ! Abfangroutine neu angeben          .
    Print "Normal-Error!"                                      |
    Resume Next         ! Mit der nächsten Zeile weiter --------'
  Endif
Return
```

ON ERROR [GOSUB] — Verzweigung bei Fehler

ON ERROR GOSUB Prozedur
ON ERROR

Verzweigt im ersten Fall zur der angegebenen Prozedur, sobald ein System- oder BASIC-Fehler auftritt. In diesem Fall wird keine Fehlermeldung ausgegeben, sondern das Error-Handling wird dem Programmierer überlassen. So ist es möglich, anhand der Fehlernummer ERR (und in V3.0: ERR$) eigene Fehlermeldungen auszugeben oder das Programm - Ihren Vorstellungen und des aktuellen Errors entsprechend - weiterverzweigen oder einfach fortfahren zu lassen (siehe RESUME).

Wurde zu einer Fehler-Routine verzweigt, schaltet der Interpreter die Fehlerbehandlung nach Abarbeiten der Prozedur wieder in den Normalmodus zurück. Soll also wieder ON ERROR GOSUB aktiviert werden, so muß in der Abfangroutine **vor** dem RESUME-Sprung erneut eine/die Fehlerroutine angegeben werden.

Die zweite Syntaxvariante schaltet den normalen Error-Modus wieder ein. Es erscheint bei Errors also wieder die übliche Fehlermeldung (in Compilaten entsprechende OPTION einsetzen!) und das Programm wird daran anschließend abgebrochen.

In V3.0 kann der Befehlsteil GOSUB weggelassen werden. Er wird vom Editor selbstständig eingefügt. Beispiel unter FATAL.

RESUME { RESU } Programm nach Error-Routine fortsetzen

RESUME => Nach Error-Routine Programmfortsetzung mit Wiederholung der fehlerhaften Zeile
RESUME NEXT => Nach Error-Routine Programmfortsetzung mit der Zeile, die der fehlerhaften Zeile folgt.
RESUME Label => Nach Error-Routine Programmfortsetzung mit der angegebenen Label-Zeile.

Bestimmmt, mit welcher Programmzeile das Programm nach Auftreten eines selbst verwalteten Fehlers (siehe ON ERROR GOSUB) fortgesetzt werden soll. RESUME ist nur innerhalb von Prozeduren zulässig. Sinnvollerweise wäre das eine Fehlerbehandlungsroutine, aber es ist auch möglich, RESUME Label als GOTO-Sprung aus einer Prozedur heraus zu mißbrauchen. GOTO selbst ist ja aus Prozeduren heraus nicht erlaubt, z.B.:

```
Label:              ! Sprung-Label
Print 111           ! PRINT irgendwas
@Routine            ! Routine aufrufen
Procedure Routine
  Resume Label      ! Sprung zum Label
Return
```

Befindet sich bei der Label-Variante das angegebene Label außerhalb der Routine, in welcher das RESUME Label steht, wird grundsätzlich der GOSUB-Sprungstapel gelöscht und alle globalen Variablen restauriert.

Nach FATAL-Errors (siehe FATAL) ist ausschließlich RESUME Label zu verwenden. RESUME NEXT und RESUME könnten in diesem Fall evtl. zum Absturz führen.

Ein Beispiel finden Sie unter FATAL.

12.5 Auskünfte

BASEPAGE — Aktuelle Basepage-Adresse liefern

Var=BASEPAGE

Jedem compilierten oder assemblierten Programm, das vom Desktop aus oder durch EXEC gestartet wird, wird vom Betriebssystem ein Basis-Informations-Block (Basepage) von 256 Byte zugewiesen, in welchem in den ersten 128 Byte grundlegende Daten zum Betrieb dieses Programms eingetragen werden.

Die übrigen 128 Byte können auf verschiedene Arten verwendet werden. Z.B. wird hier (falls vorhanden) die Kommando-Zeile des Programms abgelegt, oder es kann hier der Disk-Transfer-Buffer (siehe FSETDTA()) eingerichtet werden.

Im Interpreter-Betrieb ist dies die Basepage des Interpreters.

Offsets (BASEPAGE+'x'):

+0	Zeiger auf TPA-Start (Transient-Program-Area). In den meisten Fällen ist dies ebenfalls die Adresse der Basepage.
+4	Zeiger auf erstes Byte hinter dem TPA-Bereich. Der TPA-Bereich ist der gesamte verfügbare Speicher von der Basepage bis zum Bildschirmspeicher.
+8	Zeiger auf Textsegment-Start. Das Textsegment ist der gesamte Speicherbereich, in welchem der aktuelle Programm-Code liegt. In den meisten Fällen liegt dessen Start bei BASEPAGE+256.
+12	Textsegment-Länge. Die Länge des Textsegments ist abhängig von der Programmstruktur. Üblicherweise ist dies die Länge des vom Programm für seinen Code in Anspruch genommenen Speichers. In vielen Fällen ist der Variablenbereich (Datenspeicher = DATA-Segment) jedoch in den Programmtext integriert (z.B. GFA-BASIC).
+16	Zeiger auf DATA-Segment-Start (Datenspeicher). Üblicherweise ist das DATA-Segment der Teil eines Programms, der beim Assemblieren als Datenspeicher für die variablen Daten (Bytes/Words/Longs) deklariert wurde.
+20	DATA-Segment-Länge.
+24	Zeiger auf BSS-Segment-Start (Block-Storage-Segment). Wie der Name schon sagt, werden in diesem Bereich üblicherweise Datenblöcke abgelegt, die auf eine Verwendung im Programm warten. Dies können Bit-Muster, oder Texte sein oder auch zwischengespeicherte Transfer-Blöcke beliebiger Art.

+28	BSS-Segment-Länge. In V3.0 ist die Länge dieses Blocks 256 Bytes. In diesen 256 Bytes wird die zuletzt durch <Control><P> gepufferte Programmzeile festgehalten.
+32	Zeiger auf DiscTransferAdress (siehe FSETDTA()/FGETDTA()).
+36	Zeiger auf die Basepage des Prozesses, der das aktuelle Programm aufgerufen hat.
+40	Reserviert
+44	Zeiger auf Environment-String (siehe SHEL_ENVRN()).
+48 bis +127	Reserviert Die Verwendung dieses Bereichs ist erst bei späteren TOS-Version zu erwarten. Er kann also - wie auch der Kommando-Bereich - für eigene Zwecke eingesetzt werden. Im Gegensatz zum Kommandobereich gibt es hier keine allgemeinen Konventionen über die Art der Verwendung.
+128 bis +255	Kommandozeile/DTA etc. (siehe SHEL_READ(), FGETDTA()). Bei Kommadozeilen gilt standardgemäß das erste Byte (lPEEK(BASEPAGE+128)) als Länge des Kommandos.

CRSCOL — Aktuelle Cursor-Spalte liefern

Var=CRSCOL

Reservierte Variable, die die TOS-Cursor-Spalte enthält, in welcher sich der Cursor aktuell befindet (CRSCOL = CuRSorCOLumn).

Im Gegensatz zur Funktion POS(), die die zeilenbezogene Cursor-Position liefert, kann mit CRSCOL die Spaltenposition des Cursors auf dem Bildschirm (Hires/Midres: 1 - 80/Lowres: 1 - 40) ermittelt werden. In Verbindung mit CRSLIN bildet CRSCOL das Gegenstück zu PRINT AT(S,Z).

Ein Beispiel hierzu finden Sie unter CRSLIN.

CRSLIN — Aktuelle Cursor-Zeile liefern

Var=CRSLIN

Reservierte Variable, die die TOS-Cursor-Zeile enthält, in welcher sich der Cursor aktuell befindet (CRSLIN = CuRSorLINe). CRSLIN liegt beim Standard-Font in allen Auflösungen immer im Bereich von 1 - 25. Wenn Sie in Hires die Prozedur Sysfont (siehe ASC()) mit Font = 1 einsetzen, liegt CRSLIN im Bereich von 1 - 50.

In Verbindung mit CRSCOL bildet CRSLIN das Gegenstück zu PRINT AT(S,Z).

Beispiel 1 (Hires/Midres):

```
For I=1 To 500                  ! 500 mal
  Print "CRSCOL/CRSLIN-Test  "; ! Text ausgeben
  If Crscol>50                  ! Cursor hinter 50. Spalte?
    Print                       ! Dann neue Zeile
  Endif
  If Crslin=25                  ! Cursor in unterster Zeile
    Cls                         ! Dann Bildschirm löschen
  Endif
Next I
```

Beispiel 2 (Hires/Midres/Lowres):

```
A$="TEXT RÜCKWÄRTS"
Print At(36,1);                 ! Cursor positionieren
Do                              ! Zeilen-Schleife
  Print At(36,Crslin+1);        ! Cursor neu positionieren
  Do                            ! Spalten-Schleife
    Print At(Crscol-1,Crslin); ! Cursor 1 Spalte nach links
    Print Chr$(27);"j";         ! Position speichern
    Print Mid$(A$,(36-Crscol),1) ! Zeichen ausgeben
    Print Chr$(27);"k";         ! Cursor auf gespeicherte Position
    Exit if Crscol<22           ! Cursor vor der 22. Spalte = Exit
  Loop
  Exit if Crslin>23             ! Cursor unter 23. Zeile = Exit
Loop
```

FRE() — Freien Speicherplatz ermitteln

Var=FRE(Dummy)
Var=FRE() **(nur V3.0)**

Es wird eine Garbage-Collection (Müll-Sammlung) durchgeführt und anschließend die Größe des noch freien Speichers geliefert. Dummy ist eine Integerwert ohne Bedeutung.

Der Interpreter sorgt sich ständig um den verbleibenden Speicherplatz. Dazu werden intern die nicht mehr benötigten Variablenbereiche (z.B. gelöschte String-Variablen und Felder) ermittelt, entfernt und die benachbarten Speicherbereiche zusammengeschlossen. Diese Arbeit wird vom BASIC sporadisch erledigt. Zu erkennen ist sie manchmal an kurzen "Aussetzern", bei denen man den Eindruck bekommt, daß das Programm für eine 10tel Sekunde "stottert".

Um diese Garbage-Collection gezielt einsetzen zu können, kann FRE(x) verwendet werden, auch wenn man am verbleibenden Speicherplatz garnicht interessiert ist. Ratsam ist der Einsatz von FRE(x) vor allem direkt **vor** einer Abfrage von Variablenadressen durch

VARPTR() oder ARRPTR(), da die Adreßlage der Variablen (vor allem der Strings) durch eine zwischenzeitlich intern ausgeführte Garbage-Collection verschoben worden sein kann.

In Version V3.0 ist Dummy optional. Wird eine Leerklammer angegeben, wird vor der Speicherplatz-Ermittlung keine Garbage-Collection durchgeführt.

Beispiele hierzu finden Sie unter anderem unter CALL in der Prozedur Scroll, unter EXEC, sowie unter RCALL, BMOVE und unter BYTE{}/CARD{}/LONG{} in der Prozedur Slow.

HIMEM — Erstes Byte hinter BASIC-Speicher liefern

Var=HIMEM

HIMEM ist eine reservierte Variable, die die Adresse des ersten Bytes hinter dem vom GFA-BASIC bzw. GFA-Compilats belegten Speicher enthält.

Beispiele hierzu finden Sie unter anderem unter EXEC, BMOVE und unter BYTE{}/CARD{}/LONG{} in der Prozedur Slow.

Version 3.0

L~A — Basis-Adresse der Line-A-Variablen liefern

Var=L~A

Liefert die Basis-Adresse der Line-A-Variablen.

Da fast sämtliche von GEM und TOS benutzten Variablen (Line-A, VDI-Escapes, Mausdaten, Sprite-Definition-Block, Workstation-Infos, RGB-Einstellungen, Font-Header etc.) an feststehenden Adressen liegen, ist durch Feststellung dieser Basis-Adresse fast der gesamte Variablenbereich des Systems überschaubar.

Für die V2.xx-Versionen: die Line-A-Basisadresse liegt bei Adresse &H293A und ist konstant. Änderungen sind erst bei neueren TOS-Versionen zu erwarten. Für das MEGA-TOS (Blitter-TOS) kann ich keine Garantie übernehmen, da ich selbst nicht mit einem MEGA-ST arbeite.

DEFFN-Funktion für V2.xx:

```
Var=@L_A
  Deffn L_A=&H293A

  L~A
Offset:  Format:  Bedeutung:
 -856 -.    5     Hier liegen die ersten 5 MKI$-Words
  bis  |- Words   des aktuellen DEFMOUSE-Strings
 -844 -'
 -846 -.          Maskenmuster Zeile 1
   .   |           . Hier liegen die 32 Bit-Muster-Words des
   .   |  32       . aktuellen DEFMOUSE-Zeigers. Abwechselnd
  bis  |- Words    . Maskenmuster und Mausmuster jeder Zeile,
   .   |           . beginnend mit dem Maskenmuster der
   .   |           . ersten Zeile.
 -784 -'          Mausmuster Zeile 16

 -692 -.          Die hier liegenden 45 Words entsprechen in
   .   |          ihrer Reihenfolge exakt dem Open-Workstation-
   .   |          Array WORK_OUT() von Index 0 bis Index 44
   .   |          (siehe dort). Dies ist für diejenigen interessant,
   .   |  45      die bis zum Erscheinen des V3.0-Compilers noch
  bis  |- Words   in GFA-V2.xx arbeiten, da sie anders nicht an
   .   |          die Workstation-Informationen herankommen. Aber
   .   |          auch die V3.0-Programmierer können diese Adressen
   .   |          zum DPOKEn verwenden, da WORK_OUT() ja keine
 -604 -'          Zuweisungen annimmt.

 -602    Word     Aktuelle Maus-X-Koordinate --.
 -600    Word     Aktuelle Maus-Y-Koordinate   |- Können durch
 -598    Word     HIDEM/SHOWM-Flag             |- DPOKE auch
 -596    Word     Aktueller Mausknopf-Status --'  gesetzt werden

 -594    Word     Rot -Intensität in Farbreg. 0  (0-7)
 -592    Word     Grün-Intensität in Farbreg. 0  (0-7)
 -590    Word     Blau-Intensität in Farbreg. 0  (0-7)
   .               .
  bis              .
   .               .
 -504    Word     Rot -Intensität in Farbreg. 15 (0-7)
 -502    Word     Grün-Intensität in Farbreg. 15 (0-7)
 -500    Word     Blau-Intensität in Farbreg. 15 (0-7)

 -498 -.          Die hier liegenden 12 Words entsprechen in
   .   |  12      ihrer Reihenfolge exakt dem Open Workstation-
  bis  |- Words   Array WORK_OUT() von Index 45 bis Index 56
   .   |          (siehe dort). Beachten Sie die Anmerkungen zu
 -476 -'          -692 bis -604.

 -460    Long     Aktuelle Adresse des 8x16-Font-Headers (RAM)
 -456    Long     Aktuelle Adresse des 6x6-Font-Headers  (ROM)
 -452    Long     Aktuelle Adresse des 8x8-Font-Headers  (RAM)
 -448    Long     Ggfs. Adresse des GDOS-Font-Headers    (RAM)
 -444    Word     Anzahl der geladenen GDOS-Fonts
```

```
-322 -.
  .   |         Der Hintergrund des Mauszeigers wird bei
  .   |  16     Mausbewegung oder SHOWM in diesem Puffer
 bis  |- Longs  gespeichert (16 Zeilen a 4 Bytes) und bei
  .   |         der nächsten Bewegung oder HIDEM/SHOWM
  .   |         mit dem Inhalt dieses Puffers restauriert.
-262 -'
```

Im folgenden Block finden Sie die L~A-Offsets der VDI-Escape-Variablen (-46 bis -2). Pokes in diese Adressen haben teilweise sehr wirkungsvolle Effekte (z.B. DPOKE L~A-44,10 - dann Text eingeben).

-46	Word	Textzeichenhöhe
-44	Word	Maximale Cursor-Spalte
-42	Word	Maximale Cursor-Zeile
-40	Word	Bytes pro Textzeile (z.B. Hires: 80*16=1280)
-38	Word	Texthintergrundfarbe
-36	Word	Textfarbe
-34	Long	Startadresse der aktuellen Cursor-Position innerhalb des Video-RAMs
-30	Word	Punkt-Offset der 1. Cursor-Zeile zum oberen Rand
-28	Word	Aktuelle Cursor-Spalte (CRSCOL)
-26	Word	Aktuelle Cursor-Zeile (CRSLIN)
-24	Byte	Cursor-Blinkrate
-23	Byte	Cursor-Blinkzähler
-22	Long	Startadresse der Font-Musterdaten
-18	Word	Letzter ASCII im Font
-16	Word	Erster ASCII im Font
-14	Word	Byte-Breite einer Font-Scan-Line (8x16-Font = 256 -> 256 Zeichen * 8 Pixel je Zeichen)
-12	Word	Bildschirmbreite in Punkten (640/640/320)
-10	Long	Startadresse der Font-Offset-Tabelle
-6	Word	Cursor-Flag (an/aus)
-4	Word	Bildschirmhöhe in Punkten (400/200/200)
-2	Word	Bildschirmbreite in Bytes

Hier beginnen die eigentlichen Line-A-Variablen

+/-0	Word	Anzahl der Bit-Planes (1, 2, 4)
+2	Word	Bildschirmbreite in Bytes
+4	Long	Startadresse des VDI-CONTRL-Arrays
+8	Long	Startadresse des VDI-INTIN-Arrays
+12	Long	Startadresse des VDI-PTSIN-Arrays
+16	Long	Startadresse des VDI-INTOUT-Arrays
+20	Long	Startadresse des VDI-PTSOUT-Arrays
+24	Word	Hintergrundfarbe für Bit-Plane 1
+26	Word	Hintergrundfarbe für Bit-Plane 2
+28	Word	Hintergrundfarbe für Bit-Plane 3
+30	Word	Hintergrundfarbe für Bit-Plane 4
+32	Word	Letztes Pixel einer Linie zeichnen (ja/nein)
+34	Word	Aktuelles Linienmuster
+36	Word	Aktueller Grafikmodus (0, 1, 2, 3)
+38	Word	Koordinatenpuffer (X1)
+40	Word	Koordinatenpuffer (Y1)
+42	Word	Koordinatenpuffer (X2)
+44	Word	Koordinatenpuffer (Y2)
+46	Long	Startadresse des aktuellen Füllmusters

+50	Word	Aktuelle Füllmuster-Maske
+52	Word	Mehrfarbige Füllmuster möglich (ja/nein)
+54	Word	Clipping-Flag (an/aus)
+56	Word	Linke X-Koordinate der Clip-Box
+58	Word	Obere Y-Koordinate der Clip-Box
+60	Word	Rechte X-Koordinate der Clip-Box
+62	Word	Untere Y-Koordinate der Clip-Box
+64	Word	????
+66	Word	Vergrößerungsfaktor
+68	Word	????
+70	Word	Textstatus (standard/proportional)
+72	Word	Quell-X-Koordinate bei Line-A-Txtblt
+74	Word	Quell-Y-Koordinate bei Line-A-Txtblt
+76	Word	Ziel-X-Koordinate bei Line-A-Txtblt
+78	Word	Ziel-X-Koordinate bei Line-A-Txtblt
+80	Word	Zeichenbreite in Pixel
+82	Word	Zeichenhöhe in Pixel
+84	Long	Startadresse der Font-Musterdaten
+88	Word	Byte-Breite einer Font-Scan-Line (siehe L~A-14)
+90	Word	Aktueller Textstil
+92	Word	Maske für lightend-Text (hell)
+94	Word	Maske für italic-Text (schräg)
+96	Word	Faktor für bold-Text (fett)
+98	Word	Rechter Offset bei italic-Text
+100	Word	Linker Offset bei italic-Text
+102	Word	Skalierung ????
+104	Word	Aktueller Winkel für Textrotation
+106	Word	Aktuelle Textfarbe
+108	Long	Adresse des 1. Arbeitspuffers für Textrotation
+112	Word	Byte-Offset zum 2. Arbeitspuffer
+114	Word	Texthintergrundfarbe
+116	Word	Transparent-Flag für Raster-Copy
+118	Long	Adresse der Füll-Routine

TYPE() — Variablentyp ermitteln

Var=TYPE(Pointer)

Pointer steht für einen *-Pointer (z.B. PRINT TYPE(*Var%)). Es wird der Typ der dadurch repräsentierten Variablen geliefert.

Type:

```
-1  = Fehler aufgetreten
 0  = Realzahlvariable (Var)
 1  = String-Variable (Var$)
 2  = 4-Byte-Integervariable (Var%)
 3  = Boolesche Variable (Var!)
 4  = Real-Feldvariable (Var())
 5  = String-Feldvariable (Var$())
 6  = 4-Byte-Integer-Feldvariable (Var%())
 7  = Boolesche Feldvariable (Var!())
```

Ab hier nur für V3.0:

```
 8  = 2-Byte-Integervariable (Var&)
 9  = 1-Byte-Integervariable (Var|)
12  = 2-Byte-Integer-Feldvariable (Var&())
13  = 1-Byte-Integer-Feldvariable (Var|())
```

Beispiel:

```
A=10.1
A$="ABC"
A%=10
A!=-1
Print Type(*A)'Type(*A$)'Type(*A%)'Type(*A!)
```

Ein weiteres Beispiel finden Sie unter * (Pointer).

Version 3.0

WORK_OUT() **Open-Workstation-Rückgabe-Array**

Var=WORK_OUT(Index)

Diese Funktion stellt den Inhalt des gesamten INTOUT- und PTSOUT-Arrays zum Zeitpunkt des letzten VDI-Open_Workstation-Aufrufs (GFA-Start) zur Verfügung. Das INTOUT-Array ist dabei in Index = 0 bis Index = 44 (für V2.xx: siehe L~A-692 bis -604), und das PTSOUT-Array in Index = 45 bis Index = 56 (für V2.xx: siehe L~A-498 bis -476) abgelegt.

DEFFN-Funktion für V2.xx:

```
Var=@Work_out(Index)
Deffn Work_out(Ind%)=Lpeek(&H293A-692+194...
                     ...*Abs(Ind%>44)+(Ind% Mod 45)*2)
```

Index:

```
0  =  Breite des Bildschirms in Pixel      (640/640/320)
1  =  Höhe des Bildschirms in Pixel        (400/200/200)
2  =  Ohne Bedeutung (immer 0)
3  =  Breite eines Pixels in mm/1000
4  =  Höhe eines Pixels in mm/1000
5  =  Anzahl möglicher Schrifthöhen       (0 = Beliebig)
6  =  Anzahl möglicher Linientypen
7  =  Anzahl möglicher Linienbreiten
8  =  Anzahl möglicher Markertypen
9  =  Anzahl möglicher Markerhöhen        (0 = Beliebig)
10 =  Anzahl der verfügbaren Zeichensätze
11 =  Anzahl der Punkt-Füllmuster         (DEFFILL ,2,x)
12 =  Anzahl der Linien-Füllmuster        (DEFFILL ,3,x)
```

```
13 =  Anzahl verfügbarer Farben             (2, 4, 16)
14 =  Anzahl verfügbarer VDI-Grafik-Grundfunktionen
15--.
bis |-Liste der verfügbaren Grafikfunktionen
24--'  .
       .  Diese beiden Listen stellen ein Hilfsmittel
       .  des GEM dar und sind für den GFA-Programmierer
       .  ohne jede Bedeutung.
25--.  .
bis |-Liste der möglichen Grafik-Attribute
34--'
35 =  Farbe verfügbar:                      0 = Nein ; 1 = Ja
36 =  Textrotation verfügbar:               0 = Nein ; 1 = Ja
37 =  Füllen verfügbar:                     0 = Nein ; 1 = Ja
38 =  VDI-cell_array verfügbar:             0 = Nein ; 1 = Ja
39 =  Anzahl verfügbarer Farben     (ohne Hintergrund)
40 =  Mauszeiger-Kontrolle  (0 = Tastatur ; 1 = Maus)
41 =  Werte-Eingaben (0 = Über Tastatur ; 1 = Anders)
42 =  Auswahltasten  (0 = F1-F10 ; 1 = Andere Tasten)
43 =  Texteingabe                  (immer 1 = Tastatur)
44 =  Art der Workstation
      0  =  Ausgabegerät
      1  =  Eingabegerät
      2  =  Ein-/Ausgabegerät
      3  =  Reserviert
      4  =  Metafile-Ausgabe
45 =  Minimale Zeichenbreite      ---.P
46 =  Minimale Zeichenhöhe           |T
47 =  Maximale Zeichenbreite         |S
48 =  Maximale Zeichenhöhe           |O
49 =  Minimale Zeilenbreite          |U
50 =  Ohne Bedeutung (immer 0)       |T
51 =  Maximale Zeilenbreite          |-
52 =  Ohne Bedeutung (immer 0)       |K
53 =  Minimale Markerbreite          |o
54 =  Minimale Markerhöhe            |p
55 =  Maximale Markerbreite          |i
56 =  Maximale Markerhöhe         ---'e
```

12.6 Tastaturkontrolle

Version 3.0

KEYDEF { KEYD } — Funktionstasten belegen

KEYDEF Taste,Text

Erlaubt die Belegung der Funktionstasten (F1 - F10). Es kann ein String von max. 31 Zeichen zugeordnet werden, der dann - sowohl im Programm, als auch im Editor - durch Druck auf die entsprechende Funktionstaste an der aktuellen Cursor-Position ausgegeben wird.

Taste:

1 - 10 = F1 bis F10 ohne <Shift>-Betätigung
11 - 20 = F1 bis F10 mit <Shift>-Betätigung

Soll der zugeordnete Text im GFA-Editor ausgegeben werden, muß zusätzlich noch die <Alternate>-Taste gedrückt werden (vgl. KEYPAD Bit 4 und 5). Der Text kann auch innerhalb von Dialog- und Fileselect-Boxen ausgegeben werden.

Beispiele:

Ich bin der Meinung, daß eine Funktionstastenbelegung ohne die Möglichkeit, die zugewiesenen Strings speichern, laden und ausgeben zu können, nur die Hälfte wert ist. Innerhalb eines Programms ist dies ja kein Problem, da ja die Strings immer in irgeindeiner String-Variablen (vorzugsweise einem String-Feld) abgelegt und somit ständig verfügbar sind.

Im GFA-Editor dagegen sieht die Sache anders aus. Die Strings sind irgendwo im Interpreter gepuffert und nur dann zu sehen, wenn man eben die entsprechende Funktionstaste (zusammen mit <Alternate>) drückt.

Ich habe mich nun anhand der Prozedur Find (siehe Beispiel 2 zu BMOVE) auf die Suche nach diesen Strings im Interpreter gemacht und bin (natürlich) fündig geworden. Ob die hier verwendeten Basepage-Offsets allerdings in späteren Versionen noch gültig sind, kann ich nicht versprechen. Dann ist es allerdings kein größeres Problem, die neuen Offsets mit Find ausfindig zu machen.

```
Procedure Keylist(Flag%)
  ' Diese Prozedur gibt die KEYDEF-Strings sauber geordnet
  ' entweder auf dem Bildschirm oder dem Drucker aus und/oder
  ' füllt das String-Feld Defkey$() der Reihe nach (1-20)
  ' mit den KEYDEF-Strings. Was Sie mit dem Feld anfangen,
  ' bleibt Ihrer Phantasie überlassen.
  '
  ' Flag% =
  '        0 = Es wird nur das String-Feld Defkey$() mit den 20
  '            aktuelle KEYDEF-Strings gefüllt.
  '        1 = Es wird 0 ausgeführt und gleichzeitig die Strings
  '            auf dem Bildschirm ausgegeben.
  '        2 = Es wird 0 ausgeführt und gleichzeitig die Strings
  '            auf dem Drucker ausgegeben. Ist der Drucker nicht
  '            angeschlossen oder OffLine, so wird die Routine
  '            nicht ausgeführt.
  Local I%,Back%
  If Flag%=1                    ! Bildschirm-Ausgabe?
```

```
    Open "",#99,"CON:"          ! Dann Console öffnen
  Endif
  If Flag%<0 Or Flag%>1         ! Druckerausgabe?
    If Out?(0)                  ! Drucker OnLine?
      Open "",#99,"PRN:"        ! Dann Centronics-Port öffnen
    Else                        ! Drucker nicht empfangsbereit!
      Back%=1                   ! Abbruch-Flag setzen
    Endif
  Endif
  If Back%=0                    ! Alles okay?
    Erase Defkey$()             ! String-Feld löschen
    Dim Defkey$(20)             ! String-Feld dimensionieren
    For I%=0 To 19              ! 20 Strings
      Defkey$(I%+1)=Chr$(34)+Char{Basepage+40034+I%*32}+Chr$(34)
      '                         ! In Keydef$() einlesen
      If Flag%                  ! Ausgabe?
        Print #99;"KEY ";Spc(2-Len(Str$(I%+1)));
        Print #99;I%+1;" : ";Defkey$(I%+1) ! Dann ausgeben
      Endif
    Next I%
  Endif
  Close #99                     ! Port wieder schließen
Return
'
Procedure Keysave(Datname$)
  ' Speichert den internen Original-KEYDEF-Block in einer
  ' beliebigen Disk-Datei ab.
  ' Datname$ = Beliebiger Dateiname ohne Extension.
  '            Die Extension .KEY wird automatisch gesetzt.
  '
  If Instr(Right$(Datname$,4),".")=0 ! Keine Extension?
    Bsave Datname$+".KEY",Basepage+40034,20*32 ! Save Block
  Endif
Return
'
Procedure Keyload(Datname$)
  ' Lädt einen durch Keysave gespeicherten KEYDEF-Block
  ' aus einer beliebigen Datei mit der Extension .KEY
  ' und schreibt ihn in den internen KEYDEF-Puffer.
  ' Datname$ = Beliebiger Dateiname ohne Extension.
  '            Die Extension .KEY wird automatisch gesetzt.
  '
  If Exist(Datname$+".KEY")     ! Angegebene .KEY-Datei existiert?
    For I%=1 To 20              ! 20 KEYDEF-Strings
      Keydef I%,Space$(31)      ! Auf Länge trimmen
    Next I%
    Bload Datname$+".KEY",Basepage+40034 ! Und Block laden
  Endif
Return
'
Procedure Keyline(Fkey%)
  ' Kleines Bonbon! Diese Routine belegt die Funktionstaste
  ' Fkey% (1 - 20 s.o.) mit maximal 31 Zeichen des zuletzt
  ' durch <Control><P> (siehe 24.5 "Der Editor") gelöschten
  ' String(-Teils).
  ' Die Funktionstasten lassen sich auch im Direktmodus
  ' einsetzen und so kann man beliebige Programmzeilenteile
  ' in den Direktmodus hinüberholen (z.B. um sie dort
  ' einzeln zu testen).
```

```
    ' Fkey% = Index der gewünschten Funktionstaste (1 - 20)
    '
    If Fkey%=>1 And Fkey%<=20   ! Zulässiger Index?
      ' Keydef Fkey%,Left$(Char{Basepage+100578},31)
      '    Wenn Sie diese Zeile statt der unteren aktivieren, wird
      '    nicht der <Control><P>-Puffer gelesen, sondern die max.
      '    ersten 31 Zeichen der durch <Control><Delete> gelöschten
      '    Programmzeile.
      Keydef Fkey%,Left$(Char{{Basepage+24}},31)
    Endif
  Return
```

Version 3.0

KEYGET { KEYG } — Auf Taste warten

KEYGET Var

Dieser Befehl ist grundsätzlich identisch mit INP(2). Es wird auf einen Tastendruck gewartet und in der angegebenen numerischen Variablen Var ein 32-Bit-Wert (siehe KEYPRESS) zurückgegeben.

Bit:

0-7	ASCII-Wert der Taste.
8-15	Null.
16-23	Scan-Code der Taste.
24-31	Ggfs. Umschalttasten-Status zum Zeitpunkt der Betätigung:
24	<Shift-rechts>.
25	<Shift-links>.
26	<Control>.
27	<Alternate>.
28	<CapsLock> an/aus.
29-31	Null.

Wird eine Byte-Integervariable für Var (Var|) verwendet, werden nur Bit 0 - 7 bzw. bei Word-Integervariablen (Var&) nur Bit 0 - 15 der beschriebenen 32 Bit geliefert.

DEFFN-Funktion für V2.xx:

```
Var=@Keyget   ! Exakt dasselbe wie KEYGET
Deffn Keyget=Gemdos(7)
```

Version 3.0

KEYLOOK {KEYL} — 1. Zeichen aus Tastaturpuffer lesen

KEYLOOK Var

KEYLOOK gibt in Var einen 32-Bit-Wert zurück (s. KEYGET/ KEYPRESS), der dem vordersten Zeichen (Head) des Tastaturpuffers entspricht. Das ist das Zeichen, das beim nächsten Aufruf von INKEY$-, KEYTEST-, KEYGET-, INP(2) etc. geliefert werden würde. Der Zustand des Tastaturpuffers bleibt dadurch unverändert, das heißt, daß im Gegensatz zu allen anderen Tastaturabfragen das ermittelte Zeichen unverändert an seiner Position im Puffer stehen bleibt.

Version 3.0

KEYPAD {K} — Tastatur-Attribute definieren

KEYPAD Vektor

Durch diesen Befehl kann die Tastaturbelegung auf verschiedene Weise beeinflußt werden. Die Funktionen sind anschließend innerhalb des Programms verfügbar.

In Vektor wird dabei ein Bit-Vektor von max. 6 Bits übergeben:

Bit 0	**NumLock-Modus (PC-Ziffernblock) (0/1 = An/Aus) Siehe 24.5 "Der Editor".**
Bit 1	**NumLock-Modus (0/1 = Schaltbar/Nicht schaltbar).**
Bit 2	**Cursor-Steuerung durch <Control>+<Pfeiltaste> (0/1 = An/Aus).**
Bit 3	**Sonderzeicheneingabe durch <Alternate> und anschließender ASCII-Werteingabe (0/1 = An/Aus).**
Bit 4	**Durch KEYDEF belegte Funktionstasten sind verfügbar und können durch <Fxx> bzw. <Shift><Fxx> im Programm ausgelöst werden.**
Bit 5	**Die durch KEYDEF belegten Funktionstasten sind nur verfügbar, wenn zusätzlich zu <Fxx> bzw. <Shift><Fxx> noch <Alternate> gedrückt wird. Ist keines der beiden letzten Bits gesetzt, können die Funktionstasten-Strings nicht aufgerufen werden.**

Durch GFA-BASIC wird bei V3.0-BASIC-Start automatisch KEYPAD &X101110 (46) initialisiert. Auf dem ST gilt normalerweise die Tastaturbelegung, die KEYPAD 0 entspricht.

Bei Programmen aus den V2.xx-Versionen, die in V3.0 gestartet werden und die Tastatur außerhalb der Normal-Tastatur (Ziffern-, Cursor-Block und Funktionstasten) in Verbindung mit <Control> oder <Alternate> abfragen, muß vorher KEYPAD 0 eingestellt werden, da sonst falsche Ergebnisse geliefert werden.

Version 3.0

KEYPRESS { KEYPR } Tastendruck simulieren

KEYPRESS Code

Hiermit kann man einen Tastendruck simulieren. In Code wird dabei ein Wert beliebigen Formats (Byte, Word oder Long) übergeben. Das LO-Byte von Byte- oder Word-Werten wird dabei als ASCII-Wert der zu simulierenden Taste interpretiert. Wird ein Longword übergeben, so wird das HI-Byte des HI-Words als gewünschter Umschalttasten-Status und das LO-Byte des HI-Words als gewünschter Scan-Code gewertet (siehe KEYGET).

Die Simulation wird dann wirksam, wenn die nächste Gelegenheit zur Eingabe vorhanden ist, z.B. bei INPUT, INKEY$, INP(2), KEYGET etc., sowie in GEM-Eingabezeilen (Dialog- und Fileselect-Boxen).

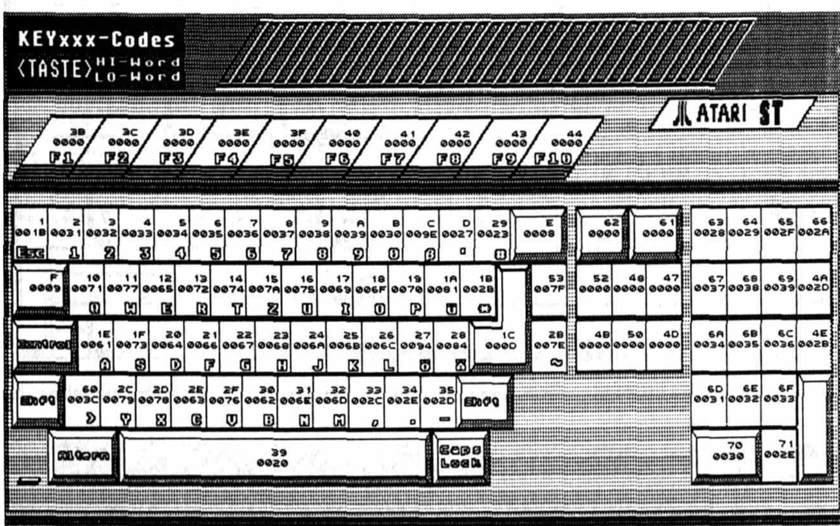
Beispiel 1:

```
A$="FILENAME.DAT"              ! File-Name
For I%=1 To 12                 ! 12 Tasten
  Keypress Asc(Mid$(A$,I%,1))  ! "drücken"
Next I%
Keypress 4718592               ! <Pfeil-hoch>-Taste
A$="\PFAD\BLABLA"              ! Pfadbezeichnung
For I%=1 To 12                 ! 12 Tasten
  Keypress Asc(Mid$(A$,I%,1))  !
Next I%
Keypress 65563                 ! <Esc>-Taste
Keypress 5242880               ! <Pfeil-runter>-Taste
For I%=1 To 12                 ! 12 mal
  Keypress 917512              ! <Backspace>
Next I%
For I%=1 To 12                 ! 12
  Keypress 34340927            ! Fragezeichen
Next I%
Keypress 4718592               ! <Pfeil-hoch>-Taste
For I%=1 To 10                 ! 10
  Keypress 34340927            ! Fragezeichen
Next I%
Fileselect "\*.*","",B$        ! Und dann los..
```

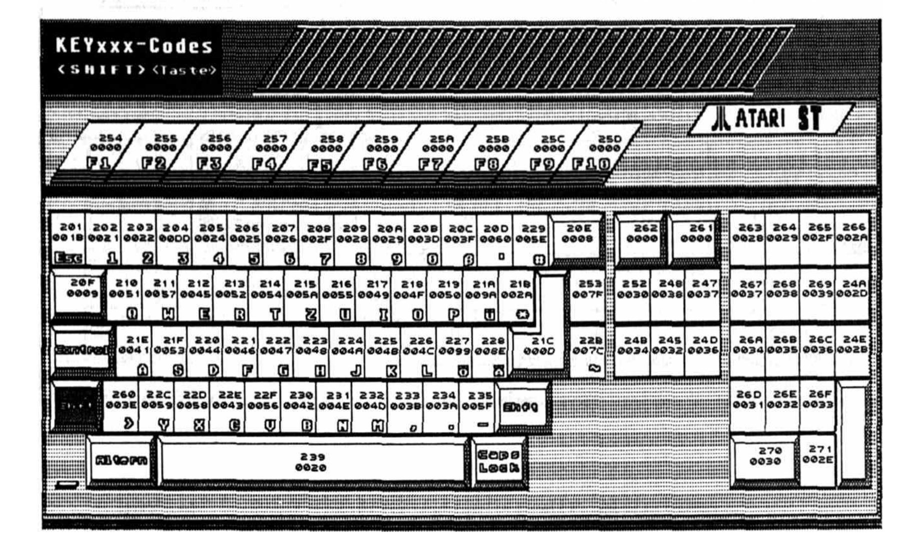
Das macht richtig Spaß, wie von Geisterhand die Fileselect-Box zu bedienen.

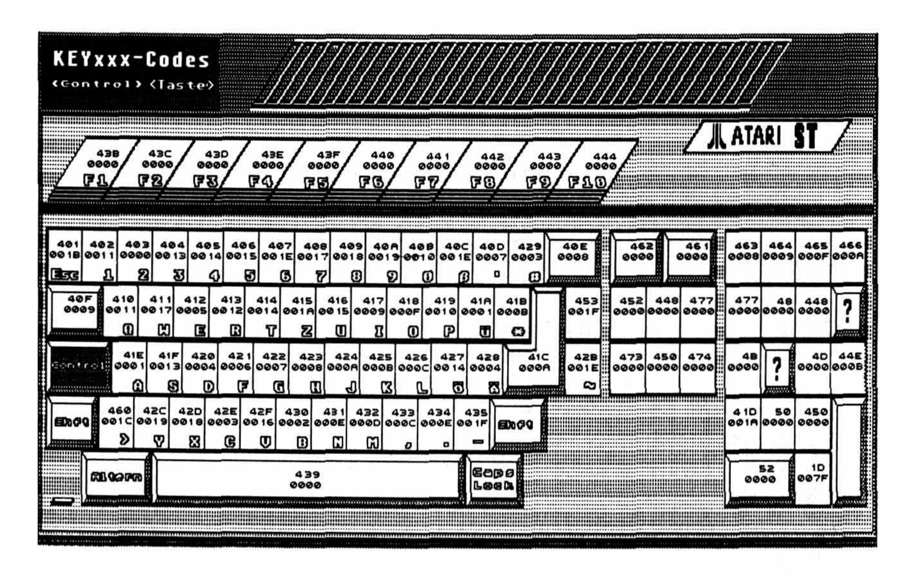
In den nachfolgenden Grafiken finden Sie alle relevanten Tastatur-Codes verzeichnet. Keypressing macht nämlich nur dann richtig Spaß, wenn man auch weiß, welche Taste man gerade drückt. Die auf die Tasten gedruckten Werte sind die 4-Byte-Werte, die jede Taste durch

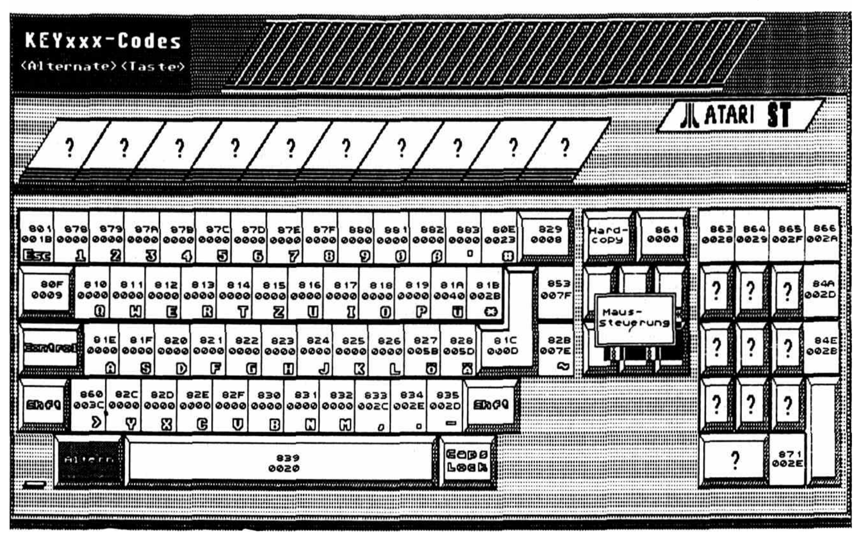
KEYGET oder KEYTEST liefert. Die obere Zahlenreihe entsprechen dem HI-Word und die untere dem LO-Word des Codes. So kann man spielend die jeweiligen Scan- und ASCII-Codes erkennen.



Erstellung der Grafik mit 'GRAF PRO'




KEYxxx-Codes
<Control> <Taste>
ATARI ST


KEYxxx-Codes
<Alternate> <Taste>
ATARI ST
Hard-copy
Maus-Steuerung

Version 3.0

KEYTEST { KEYT } Tastatur abfragen

KEYTEST Var

Dieser Befehl ist grundsätzlich identisch mit INKEY$, nur daß er nicht wie diese als Funktion innerhalb von Ausdrücken verwendet werden kann.

Es wird in Var ein 32-Bit-Wert (siehe KEYGET/KEYPRESS) zurückgegeben, welcher der bei Befehlsausführung gedrückten Taste entspricht. Es wird nicht auf den Tastendruck gewartet, sondern im "Vorübergehen" nachgeschaut, ob eine Taste gedrückt wurde. Ist das nicht der Fall, erhält man den Wert 0.

DEFFN-Funktion für V2.xx:

```
Var=@Keytest  ! Exakt dasselbe wie KEYTEST
Deffn Keytest=Gemdos(6,255)
```

12.7 Multitasking

Version 3.0

AFTER x GOSUB { AF } Single-Interrupt-Routinenaufruf

AFTER Ticks [GOSUB] Prozedur

Ruft nach Ticks Zeiteinheiten (1 Sekunde = 200 Ticks) die angegebene Procedur **einmal** auf. Der Befehl wird jedoch - generell - immer erst nach vollständiger Abarbeitung eines BASIC-Befehls ausgeführt. Wenn z.B. durch SOUND x,x,x,x,Dauer, PAUSE, DELAY, INP() INPUT, BLOAD etc. die Programmausführung unterbrochen wird, so wird der Sprung zur angegebenen Prozedur ggfs. erst bei Wiederaufnahme des Programms wirksam.

Beispiel:

```
After 100 Gosub Abc  ! AFTER-Aufruf
T%=Timer
Input A$             ! AFTER-Ausführung wird unterbrochen
Do
Loop
Procedure Abc
  Print "AFTER 100 GOSUB erst nach ";Timer-T%;
  Print " Ticks ausgeführt
```

```
  End
Return
```

Anmerkung: Für die Interrupt-Verwaltung wird der Event-Timer-Vektor (etv_timer) des GEM bei Adresse &H400 (Supervisor) verwendet. Es ist deshalb bisher nicht möglich, mehrere AFTER-Interrupt-Anweisungen gleichzeitig zu verarbeiten. Die zuletzt verfügte AFTER GOSUB- bzw. EVERY GOSUB-Anweisung hebt also die vorangegangene auf.

Beispiel:

```
After 1000 Gosub Aa   ! Erster Aufruf
After 400 Gosub Bb    ! Zweiter Aufruf
Do                    ! Endlos-Schleife
Loop
Procedure Aa          ! Prozedur für ersten Aufruf
  Print 11111
Return
Procedure Bb          ! Prozedur für zweiten Aufruf
  Print 22222
Return
```

Hier im Beispiel wird nur einmal nach 2 Sekunden die Prozedur des zweiten Aufrufs angesprungen. Der erste Aufruf ist durch den zweiten ungültig geworden. PRINT 11111 wird also nicht mehr ausgeführt.

Es gibt allerdings die Möglichkeit, aus einer AFTER x GOSUB-Anweisung eine zweite EVERY x GOSUB-Anweisung zu machen. Dazu muß ganz einfach die entsprechende AFTER x GOSUB-Prozedur vor ihrem RETURN wieder neu initialisiert werden. Dabei ist zu beachten, daß so nicht der gleiche regelmäßige Rythmus wie bei EVERY zu erreichen ist, da ja die Initialisierung selbst immer wieder Zeit in Anspruch nimmt. Eine solche - zu EVERY zusätzliche - Interrupt-Schleife ist jedoch in den meisten Fällen besser als keine.

Beispiel:

```
After 60 Gosub Proc1        ! AFTER initialisieren
Every 100 Gosub Proc2       ! EVERY initialisieren
Do                          ! Endlos-Schleife
Loop
Procedure Proc1             ! AFTER-Prozedur
  Print "AFTER-Procedure"
  After 60 Gosub Proc1      ! AFTER neu initialisieren
Return
Procedure Proc2             ! EVERY-Prozedur
  Print "EVERY-Procedure"
Return
```

Version 3.0

AFTER CONT { AF CONT } Single-Interruptroutine freigeben

AFTER CONT

Setzt nach AFTER STOP die Single-Interruptkontrolle fort. Weiteres siehe AFTER STOP.

Version 3.0

AFTER STOP { AF STOP } Single-Interruptroutine sperren

AFTER STOP

Unterbricht die Single-Interruptkontrolle.

Durch AFTER STOP kann veranlaßt werden, daß die Ausführung der aktuellen AFTER x GOSUB-Prozedur solange verzögert wird, bis wieder AFTER CONT verfügt wird. Wird AFTER CONT nicht eingesetzt, so bleibt AFTER x GOSUB bis zum Programmende inaktiv - es sei denn, durch einen neuen AFTER x GOSUB wurde ein neuer Timer gesetzt.

Z.B. soll nach 3 Sekunden eine AFTER-Procedure angesprungen werden. Die Ausführung wird jedoch nach 2 Sekunden durch AFTER STOP unterbrochen und nach weiteren 2 Sekunden durch AFTER CONT wieder zugelassen, so wird AFTER x GOSUB erst nach diesen 4 Sekunden ausgeführt, obwohl die Zeitspanne von 3 Sekunden schon verstrichen ist. Weitere AFTER CONT-Aufrufe für denselben Prozeß haben dann keine Wirkung mehr, da die Routine nur **einmal** ausgeführt wird.

Beispiel:

```
T%=Timer
After 600 Gosub Proc1     ! Nach 3 Sekunden
Print "AFTER 600 GOSUB ist aktiv"
Every 400 Gosub Proc2     ! Alle 2 Sekunden
Do                        ! Endlos-Schleife
Loop
Procedure Proc1
  Print "AFTER 600 GOSUB nach ";(Timer-T%);" Ticks ausgeführt!"
  Edit
Return
Procedure Proc2
  A%=A% Xor 1             ! A% als Wechsel-Flag (1/0)
  If A%=1
    Print "AFTER 600 GOSUB unterbrochen"
```

```
    After Stop          ! AFTER GOSUB unterbrechen
  Else
    Print "AFTER 600 GOSUB wieder aufgenommen"
    After  Cont         ! AFTER GOSUB fortsetzen
  Endif
Return
```

EVERY x GOSUB { EV } — Interrupt-Routinenaufruf

EVERY Ticks [GOSUB] Prozedur

Ruft **ständig** nach Ablauf von Ticks Zeiteinheiten (1 Sekunde = 200 Ticks) die angegebene Prozedur auf. Der Befehl wird jedoch - generell - immer erst nach vollständiger Abarbeitung eines BASIC-Befehls ausgeführt. Wenn z.B. durch SOUND x,x,x,x,Dauer, PAUSE, DELAY, INP() INPUT, BLOAD etc. die Programmausführung unterbrochen wird, so wird der Sprung zur angegebenen Prozedur ggfs. erst bei Wiederaufnahme des Programms wirksam.

Die GFA-Fans mußten lange auf diesen Befehl warten. Beachten Sie außerdem die Anmerkung zu AFTER GOSUB.

Beispiel:

```
Every 50 Gosub Multi        ! Interrupt initialisieren
Print At(1,3);"Bitte Text eingeben und mit Maus zeichnen."
Print "(Die Zeit/Datum-Anzeige kann durch Klick";
Print "geöffnet und dann geändert werden.)"
Do                          ! Endlos-Schleife
  Keytest A%                ! Tastatur abfragen
  If (A% And 255)           ! ASCII-Taste gedrückt?
    Out 5,A% And 255        ! ASCII über Console ausgeben
  Endif
  If Mousek                 ! Maustaste gedrückt?
    Repeat                  ! Box-Schleife
      Box Mousex,Mousey,Mousex+10,Mousey+10
    Until Mousek=0          ! Bis Maustaste losgelassen
  Endif
Loop
Procedure Multi             ! EVERY-Prozedur
  Every Stop                ! EVERY unterbrechen (siehe unten)
  Gosub Timdat(1,1,1)       ! Zeit-Prozedur aufrufen
  Every  Cont               ! EVERY fortsetzen   (siehe unten)
Return
```

Die in diesem Beispiel verwendete Zeit-Routine Timdat finden Sie unter SETTIME beschrieben. Sie ist in dieses Demo-Programm vor Start einzubinden.

Hier können Sie einen Effekt beobachten, der mich fast zur Verzweiflung getrieben hat. Ich wußte genau, daß die Timdat- Routine einwandfrei funktionierte. Als ich den Timdat-Aufruf in die die Multi-Prozedur blauäugig eingebunden hatte, war ein Probelauf fällig ---!!!--- und es funktionierte garnichts mehr ---!!!---!!! Immer, wenn ich das Datum oder die Zeit ändern wollte, brach das Chaos aus.

Ich durchsuchte Timdat nach möglichen Fehlern und suchte und suchte... Als ich dann genug gesucht hatte, überprüfte ich mit TRON den Programmlauf. Da die Interrupt-Prozeduren von TRON ausgeschlossen werden (!), konnte ich so auch nichts erkennen. Nach einiger - haareraufend verbrachter - Zeit kam mir der Geistesblitz, daß ja die Timdat-Routine auch dann immer wieder durch EVERY aufgerufen wird, wenn man innerhalb von Timdat gerade mit der Zeit- oder Datumseingabe beschäftigt ist. Daraus entstand - quasi rekursiv - das Durcheinander. Erst als ich vor dem Timdat-Aufruf EVERY STOP und danach EVERY CONT einsetzte, funktionierte alles wieder so, wie es sollte.

Version 3.0

EVERY CONT { EV CONT } — Interrupt-Routine freigeben

EVERY CONT

Nimmt nach EVERY STOP die EVERY-Interruptkontrolle wieder auf. EVERY GOSUB wird wieder ausgeführt.

Version 3.0

EVERY STOP { EV STOP } — Interrupt-Routine sperren

EVERY STOP

Unterbricht die EVERY-Interruptkontrolle. EVERY GOSUB wird nicht mehr ausgeführt. Fortsetzung erst wieder durch EVERY CONT.

12.8 Debugging

Version 3.0

DUMP { DU } Variableninhalte/Namen ausgeben

DUMP [Defstring$] [TO Datei$]

Wird DUMP ohne Optionen verwendet, werden die Inhalte aller Variablen, sowie die Dimensionierungen aller Felder ausgegeben.

In Deftsring$ kann optional eine nähere Spezifikation angegeben werden angegeben werden:

"a"	Die Inhalte aller Variablen, sowie die Dimensionierungen aller Felder, deren Name mit a beginnt, werden ausgegeben.
":"	Alle Label-Namen, sowie ihre Zeilennummer werden ausgegeben.
":a"	Alle Label-Namen, die mit a beginnen, sowie ihre Zeilennummer werden ausgegeben.
"@"	Alle Prozedur- und Funktionsnamen, sowie ihre Zeilennummer werden ausgegeben.
"@a"	Alle Prozedur- und Funktionsnamen, die mit a beginnen, sowie ihre Zeilennummer werden ausgegeben.

Ausgegebenen Prozedurnamen wird als Kennung @ angehängt, sowie dahinter - falls die Prozedur im aktuellen Programm noch existiert - die Nummer der Zeile, in welcher die Prozedur beginnt. Dasselbe geschieht bei Namen noch existierender Funktionen. Diese erhalten jedoch als Kennung ein FN. Die Namen von String-Funktionen werden zusätzlich noch mit einem $ gekennzeichnet.

Prozeduren, Funktionen und Label, die definiert waren und wieder aus dem Programm gelöscht wurden, werden ohne Zeilennummern dargestellt. Dagegen sind nicht mehr existierende Variablennamen - die ebenfalls ausgegeben werden, solange sie noch in der internen Liste stehen - auf den ersten Blick nicht zu erkennen. Das einzige Merkmal ist, daß sie keinen Inhalt haben (0 bzw. ""), was jedoch bei noch existierenden Variablen ebenso der Fall sein kein.

Durch Save,A, New und anschließendes Merge werden die überflüssigen Namen aus der internen Referenzliste entfernt (siehe 24.5 "Der Editor").

Bei String-Variablen werden max. 60 Zeichen ihres Inhalts ausgegeben, die dann in " (Anführungszeichen) eingeschlossen sind. Ist der Inhalt länger als 60 Zeichen, steht am Textende das Zeichen >. Enthält

der String Zeichen mit einem ASCII-Wert unter 32, werden diese Zeichen durch einen Punkt . repräsentiert.

Es kann außerdem - unabhängig von der Option Deftsring$ - hinter dem Zusatz TO in Datei$ der Name einer Datei angegeben werden, in welche dann die DUMP-Ausgaben umgeleitet werden. Diese Datei erhält- sofern keine andere angegeben wird - die Extension .DMP. Ggfs schon existierende .DMP-Dateien gleichen Namens erhalten die Backup-Extension .BAK.

Auch hier können, wie unter OPEN beschrieben, durch

```
TO "VID:" ... TO "CON:" ... TO "LST:" ... etc.
```

die einzelnen Ports angesprochen werden (weiteres siehe unter OPEN).

Statt TO kann wahlweise auch ein Komma verwendet werden, das der Interpreter dann automatisch in TO umwandelt. Ein Beispiel zu DUMP finden Sie unter TRON Proc.

Da es in den V2.xx-Versionen kein DUMP gibt, folgt hier ein kleines Programm, das Ihnen die Variablen-, Label-, Prozedur- und Funktionsnamen eines .BAS-Programms von der Diskette liefert. Genau wie bei DUMP, erhalten Sie eine - fein säuberlich nach Typen getrennte - Liste.

Den meisten von Ihnen wird bekannt sein, daß jedes .BAS-File bzw. auch jedes V3.0-.GFA-File über einen Vorspann verfügt, in welchem mehrere wichtige Daten vom Interpreter. abgelegt werden. Nur wenigen wird bekannt sein, wie dieser Header aufgebaut ist. Machen Sie sich keine Hoffnung, ich werde es Ihnen auch nicht verraten. Aus einem einfachen Grund. Mit genauen Kenntnissen über den Aufbau des Headers wäre es spielend möglich, mit PSAVE geschützte Programme zu "knacken". Da ich nicht davon ausgehe, daß schon jeder über einen GFA-BASIC-Compiler verfügt, muß diese Möglichkeit der Programmsicherung erhalten bleiben.

Was allerdings kein Geheimnis ist, ist die Organisation des Bereichs, in dem das BASIC die verwendeten Variablen-, Funktions-, Label- und Prozedurnamen abspeichert. Mit diesen Kenntnissen ist es in den V2.xx-Versionen dann wenigstens möglich, sich relativ leicht eine Referenzliste über die verwendeten Namen zu erstellen. Ich habe im Listing auch die entsprechenden Zeilen für die V3.0-Version eingebaut. Sie sind in REM-Zeilen verpackt und müssen nur anstatt der

V2.xx-Zeilen aktiviert werden. Bei den DATAs ist dagegen nur die DATA-Liste durch die V3.0-Zeilen zu ergänzen. Im folgenden habe ich die V3.0-relevanten Angaben in Klammern hinter die V2.xx-Angaben gesetzt.

Ab dem 10. Byte jeder BAS-Datei (GFA-Datei) liegen 13 (17) Longwords, die der Reihe nach die Offsets der einzelnen Variablentypen-Blöcke zum Byte 126 (164) der Datei enthalten. Ab Byte 126 (164) werden nämlich 12 (16) Blöcke gespeichert, die die Variablen-, Label-, Prozedur- und Funktionsnamen enthalten. In den Bytes 11 bis 14 steht also der erste Offset, der in jedem Fall 0 ist, da ja der Offset sich auf Byte 126 (164) bezieht und der erste Block bei Byte 126 (164) beginnt. Dieser erste Block enthält alle Real-Variablennamen, die im Programm verwendet wurden.

Daran schließen sich die Offsets für String-, Integer- und Boole-Variablennamen, Real-, String-, Integer- und Boole-Feldnamen, (Word- und Byte-Variablennamen) Label- und Prozedurnamen, (Word- und Byte-Feldnamen) Arithmetik- und String-Funktionsnamen an. Das 13. (17.) und letzte Longword dieser Reihe enthält den Offset vom 126. (164.) Byte zum ersten Byte des Programm-Listings.

Das war es eigentlich schon. Mit diesen Informationen lassen sich nun alle Namen feststellen. Da ist nur noch das eine Problem: nämlich die Längen der einzelnen Namen, ohne die natürlich nicht viel auszurichten ist. Die Länge jedes Variablennamens steht in einem einzelnen Byte jeweils genau vor dem Namen.

Das folgende Programm schreibt nun alle gefundenen Namen als REM-Zeilen in eine Datei mit dem Namen VARS.LST. Von dort kann diese Liste mit Merge in den Arbeitsspeicher geladen oder weiterverarbeitet werden.

Der Ausbau dieses Programms zu einem Programm, das eine echte Referenzliste aller verwendeten Variablen (evtl. mit Zeilennummern oder nach Prozeduren sortiert) enthält, dürfte keinen allzugroßen Aufwand mehr darstellen.

```
DIM b.len%(12),vtp$(12)          ! V2.xx: Feld für Blocklängen und Typen
' DIM b.len%(16),vtp$(16)        ! V3.0 : Feld für Blocklängen und Typen
FILESELECT "*.BAS","",s$         ! V2.xx: BAS-Programm wählen
' FILESELECT "*.GFA","",s$       ! V3.0 : BAS-Programm wählen
IF EXIST(s$)                     ! Datei existiert?
  OPEN "I",#1,s$                 ! Datei öffnen
  SEEK #1,12                     ! File-Pointer auf Offset-Tabelle
  FOR i%=0 TO 11                 ! V2.xx: 12 Offsets
```

```
    ' FOR i%=0 TO 15            ! V3.0 : 16 Offsets
    READ vtp$(i%)               ! Typenbezeichnung lesen
    BGET #1,VARPTR(b.len%(i%)),4 ! Block-Offset einlesen
  NEXT i%
  FOR i%=1 TO 12                ! V2.xx: 12 Blöcke
    ' FOR i%=1 TO 16            ! V3.0 : 16 Blöcke
    IF b.len%(i%)>b.len%(i%-1)  !Block belegt?
      SEEK #1,126+b.len%(i%-1)  !V2.xx: File-Pointer auf Blockstart
      ' SEEK #1,164+b.len%(i%-1)!V3.0 : File-Pointer auf Blockstart
      titel$=LEFT$(vtp$(i%-1),LEN(vtp$(i%-1))-10) ! Blocktitel
      vblk$=vblk$+"' "+CHR$(13)+"' "+titel$+CHR$(13) !-| Titelzeile
      vblk$=vblk$+"' "+STRING$(30,"=")+CHR$(13)      !-|   bilden
      FOR j%=1 TO b.len%(i%)-b.len%(i%-1) ! Blocklänge durchgehen
        byte%=INP(#1)           ! Namenlänge einlesen
        ADD j%,byte%            ! Zeichen-Zähler
        buff$=SPACE$(byte%)     ! Puffer vorbereiten
        BGET #1,VARPTR(buff$),byte% ! Namen einlesen
        EXIT IF LEN(vblk$)+LEN(RIGHT$(vtp$(i%-1),10)+CHR$(13))>32767
        ' Exit, wenn maximale String-Länge überschritten
        IF buff$>""             ! Name gültig?
          IF LEFT$(titel$,3)<>"Pro" ! Kein Prozedurblock?
            ' vblk$=vblk$+"'    "+buff$+TRIM$...
             ...(RIGHT$(vtp$(i%-1),10))+CHR$(13) ! V3.0 : Zeile bilden
            vblk$=vblk$+"'    "+buff$+RIGHT$(vtp$(i%-1),10)+CHR$(13)
            '                   ! V2.xx: Namen+Kennzeichnung einbinden
          ELSE                  ! Prozedurblock?
            vblk$=vblk$+"'    "+RIGHT$(vtp$(i%-1),9)+" "+buff$+CHR$(13)
            '                   ! Kennzeichnung+Namen einbinden
          ENDIF
        ENDIF
      NEXT j%                   ! Nächster Name
    ENDIF
    EXIT IF LEN(vblk$)+LEN(RIGHT$(vtp$(i%-1),10)+CHR$(13))>32767
    ' Exit, wenn maximale String-Länge überschritten
  NEXT i%                       ! Nächster Block
  CLOSE
  OPEN "O",#1,"VARS.LST"        ! .LST-File öffnen
  PRINT #1;vblk$                ! REM-Zeilenpuffer schreiben
  CLOSE
 ENDIF
EDIT
DATA "Real-Variablen          "
DATA "String-Variablen $        "
DATA "Integer-Variablen %        "
DATA "Boole-Variablen !        "
DATA "Real-Felder ()        "
DATA "String-Felder $()        "
DATA "Integer-Felder %()        "
DATA "Boole-Felder !()        "
'    DATA "Word-Variablen &          "
'    DATA "Byte-Variablen |          "
DATA "Labels :        "
DATA "Prozeduren  Procedure"
'    DATA "Word-Felder &()        "
'    DATA "Byte-Felder |()        "
DATA "Numerische Funktionen (FN)       "
DATA "String-Funktionen $ (FN$)   "
```

Version 3.0

TRACE$ Im Trace-Modus aktuelle Befehlszeile liefern

Var$=TRACE$

Innerhalb der durch TRON Proc bestimmten Prozedur kann durch die reservierte Variable TRACE$ der Text der aktuellen Programmzeile **vor** ihrer Ausführung ermittelt werden. Außerhalb dieser Prozedur ist TRACE$ ohne Inhalt.

Ein Beispiel zu TRACE$ finden Sie unter TRON Proc.

TROFF { TROF } Trace-Modus ausschalten

TROFF

Im Anschluß an TROFF ist das Programm wieder im normalen Ausführungsmodus. TRON - sowie TRON Proc in Version V3.0 - werden ausgeschaltet. TROFF innerhalb einer TRON Proc-Routine hat keine Wirkung.

TRON { TR } Trace-Modus einschalten

TRON [#Kanal]

TRON kann an jeder beliebigen Programmstelle eingesetzt werden und gibt ab dann während des Programmlaufs die jeweils aktuelle Programmzeile aus. Der Programmablauf wird sozusagen protokolliert.

Mit der Option #Kanal (siehe OPEN) wird in eine geöffnete Datei geschrieben, sonst auf dem Bildschirm.

Beispiel:

```
Open "O",#99,"TRONFILE.LST"
Tron #99
  -   Ab hier werden alle nachfolgend ausgeführten
  -   Programmzeilen in die Datei TRONFILE.LST
  -   geschrieben. Da die TRON-Zeilen im ASCII-
  -   Format Merge-fähig ausgegeben werden, kann
  -   man diese Protokoll-Datei ohne weiteres in
  -   den Programmspeicher laden.
Troff
  -  Ab hier wird das Programm ohne Ausgabe der
  -  Zeilen fortgesetzt.
```

Version 3.0

TRON Proc { TR } Trace-Modus in Prozedur lenken

TRON Prozedurname

Dies ist eine V3.0-Variante des bekannten TRON-Befehls.

Sie lenkt den Trace-Modus in die durch Prozedurname angegebene Prozedur um. Dabei wird nicht - wie bei TRON - automatisch die aktuelle Befehlszeile ausgegeben, sondern es kann beliebig auf die durch TRACE$ ermittelte, als nächstes auszuführende Befehlszeile oder ggfs. auf die durch DUMP ermittelten Variableninhalte reagiert werden.

Auch TRON Proc wird ggfs. durch TROFF deaktiviert. Von TRON und TRON Proc hat immer der jeweils zuletzt verfügte Befehl Priorität. TRON hebt also TRON Proc auf - und umgekehrt.

Möchten Sie eine Debugging-Routine selbst schreiben, so kann Ihnen TRON Proc dabei nicht helfen, denn sowohl die Interrupt-Routinen AFTER x GOSUB und EVERY x GOSUB, als auch die hier angegebene Debugging-Prozedur werden von der TRACE$-Ausgabe ausgeschlossen.

Beispiel

Die folgenden beiden Prozeduren sind etwas umfangreicher (die zweite mehr als die erste), aber sie haben es auch in sich. Die erste Prozedur mitsamt des kleinen Demo-Programms dient hier eigentlich nur als Vordergrund-Programm für die TRON-Proc-Routine. Dieses "Hilfs"-Programm soll hier nicht näher erläutert werden. Was es kann und wie man damit umgeht, ergibt sich aus der Demonstration und aus dem Programm-Kommentar.

Bei der TRON-Proc-Routine komme ich jedoch nicht um eine Bedienungsanleitung herum. Die Prozdur selber ist ausführlich kommentiert, wobei es jedoch in der gegebenen Kürze nicht möglich ist, jede Einzelheit zu erklären. Dazu ist die Routine zu komplex. Damit jene, die sich ernsthaft mit der Programmierung dieser Trace-Prozedur auseinandersetzen möchten, nicht zu kurz kommen, folgt dem Listing eine Aufstellung der verwendeten Variablen und ihrer Bedeutungen. Diese Routine wurde übrigens hochoptimiert in einer V3.0-typischen Struktur geschrieben. Sie ist wohl hier im Buch neben der AES-Objekt-

Programmierung das treffendste Beipiel für die Programmierung in V3.0. Die Prozedur ist nicht leicht zu durchschauen, in V2.xx wäre sie jedoch um ein vielfaches komplizierter.

Sie ist übrigens in Hires, Midres **und** Lowres lauffähig. Ein Umstand, den viele Programmierer nicht bedenken bzw. nicht beherrschen. Es ist meist relativ einfach, Hires-Programme nach Midres zu übertragen, da hauptsächlich die Y-Koordinaten und Texthöhen etc. einfach nur halbiert und die Farbmöglichkeiten berücksichtigt werden müssen. Eine Übertragung nach Lowres ist dagegen aus vielen Gründen äußerst kompliziert, da es hier nicht mehr reicht, einfach die X-Koordinaten zu halbieren. In diesem Fall müssen dermaßen viele Kleinigkeiten bedacht werden (Textlängen, Texthöhen, max. Zeilenlänge, Bit-Planes, Farben etc.), daß die meisten Programmierer davor schlicht resignieren.

Utilities dagegen - vor allem für eine Programmiersprache - sollten dagegen in allen Stufen lauffähig sein. Die Erfahrung zeigt, daß dies mit einiger Übung doch machbar ist.

Bedienungsanleitung

Die Prozedur läuft im Hintergrund mit. Merklich ist das daran, daß das Hauptprogramm sichtlich langsamer abläuft, als es unter normalen Umständen der Fall wäre.

Die Trace-Routine wird aufgerufen, indem die <Control>- in Verbindung mit der linken <Shift>-Taste gedrückt wird. Das Programm stoppt dann ggfs. solange, bis diese Tastenkombination wieder losgelassen wird.

Es erscheint am unteren Bildrand die aktuell bearbeitete Befehlszeile, **bevor** sie ausgeführt wird. Beim ersten Aufruf erfolgt diese Ausgabe so schnell, daß nichts zu erkennen ist. Das braucht Sie nicht zu ärgern, denn jetzt geht es erst richtig los. Solange die Ausgabe "in Bewegung ist" können Sie über die vier <Pfeil>-Tasten, die <Undo>- sowie die <ClrHome>-Taste in Verbindung mit der linken (!) <Shift>-Taste folgende Aktionen auslösen:

<Shift><Pfeil-links>
Der Ausgabebereich wird um eine Zeile nach oben bewegt.

<Shift><Pfeil-rechts>
Der Ausgabebereich wird um eine Zeile nach unten bewegt.

<Shift><Pfeil-hoch>
: **Ist die unterste Ausgabezeile unterhalb der 6. Cursor-Zeile, wird der Ausgabebereich um eine Zeile erweitert.**

<Shift><Pfeil-hoch>
: **Ist der Ausgabebereich höher als eine Zeile, so wird er um eine Zeile vermindert.**

Bei diesen vier Aktionen wird der Originalhintergrund des Programms weitestgehend restauriert. Bildschirmausgaben des Hauptprogramms innerhalb des geöffneten Debugger-Fensters können dabei jedoch nicht berücksichtigt werden.

<Shift><ClrHome>
: **Pausen-Einstellung. Bei jeder Betätigung dieser Kombination wird der PAUSE-Wert um 2 erhöht. Wird der Wert 20 überschritten, beginnt die Zählung wieder bei Null. Diese Pause ist nur wirksam, solange die Ausgabe geöffnet ist.**

<Shift><Delete>
: **Schaltet die Variablen-Verfolgung (Follow) abwechselnd an und aus. Ist die Verfolgung angeschaltet, so wird das Programm immer dann in den Einzelschrittmodus geschaltet, sobald der am Prozedur-Anfang angegebene Variablenname in TRACE$ enthalten ist. Das kann natürlich auch der Fall sein, wenn die Variable in Ausdrücken enthalten ist, aber ihr Inhalt nicht verändert wird. Um eine Programmunterbrechung nur dann zuzulassen, wenn dieser Variablen etwas zugewiesen wird, ist als Suchbegriff die entsprechende Zuweisungsform anzugeben (z.B. Var= oder ADD Var). Wie auch bei der unten beschriebenen Sequenz-Suche ist hier die Form der Schreibweise unerheblich.**

Im Falle, daß die angegebene Variable in TRACE$ enthalten ist, wird bei numerischen Variablen neben der angezeigten - als nächstes auszuführenden - Programmzeile der Inhalt der Variablen am rechten Bildrand angezeigt. Bei String-Variablen werden max. die ersten 74 (in Lowres 34) Zeichen des Inhalts angezeigt.

Danach wartet der Debugger auf einen Tastendruck, woraufhin die als nächstes auszuführende Zeile angezeigt wird. In beiden Fällen wird der Variableninhalt vor der angegebenen Zeile ausgegeben. Sie können die nächste Zeile nun durch <Undo> ausführen lassen und daran anschließend in den Direktmodus wechseln, um dort den neuen Inhalt zu erfragen oder sich über den DUMP-Menüpunkt (<Shift><Insert>) den Inhalt anzeigen lassen.

Damit die Prozedur den Variableninhalt ermitteln kann, müssen Sie die gewünschte Variable vor Programmstart in das Debugger-Listing einbinden (siehe die beiden mit **FOLLOW markierten Prozedurteile).**

<Shift><Undo>
: **Step-Modus.**

Beachten Sie, daß die oben genannten Debugger-Funktionen nur bei laufender Anzeige möglich sind. Im letztgenannten Step-Modus sind dieses Funktionen außer Kraft. In diesem Modus bleibt die Ausgabe und das Programm stehen und Sie haben nun folgende Möglichkeiten:

<Undo> Nächste Programmzeile ausführen.

<Shift><Undo>
Step-Modus aus. Weiter mit laufender Zeilenanzeige.

<Shift><Help>
Das Programm wird durch STOP unterbrochen. Sie befinden sich anschließend in der Direkt-Eingabeebene, von wo aus Sie auch in den Programm-Editor wechseln können (<Esc><Return>). Soll das Programm bei der aktuellen Befehlszeile fortgesetzt werden, geben Sie im Direktmodus CONT ein.

Bedenken Sie dabei, daß durch Änderungen im Listing sämtliche Variablen gelöscht und offene Dateien geschlossen werden.

<Shift><Insert>
Es erscheint ein Menü.

Dieses Menü hat vier Auswahl-Punkte:

Sequenz Ermöglicht die Eingabe einer Zeilen-Sequenz. Sobald eine Zeile ausgeführt wird, in deren Text die angegebene Sequenz enthalten ist, schaltet die Trace-Routine automatisch in den Step-Modus, auch wenn die Zeilenanzeige geschlossen ist.

Ist die Routine schon im Step-Modus, geschieht vorerst nichts. Diese Suchvorgabe eignet sich vor allem dazu, einzelne Programmzeilen zu überwachen. Da in TRACE$ der Text der als nächstes auszuführenden Programmzeile steht, wird so erreicht, daß die zu überwachende Zeile vor Ausführung angezeigt wird, um dann mit dem Einzelschritt-Verfahren beobachten zu können, was diese Zeile bewirkt.

Die zweite Möglichkeit besteht darin, einen oder mehrere "Breakpoints" in das Programm einzubauen (z.B. als Label: Breakpoint_1:), so daß das Programm immer dann unterbrochen wird, sobald es diese(n) Breakpoint(s) erreicht hat. Wird die Sequenz-Abfrage ohne Eingabe einfach durch <Return> quittiert, ist die Sequenz-Überwachung ausgeschaltet. Die Form der Schreibweise (Groß/Klein) ist bei Angabe der Sequenz unerheblich, da sowohl TRACE$ als auch die Sequenz vor der Überprüfung durch INSTR von der Funktion UPPER$ in Großschrift umgewandelt werden.

DUMP Erwartet die Eingabe eines 2 Zeichen langen DUMP-Parameter-Strings (siehe DUMP).

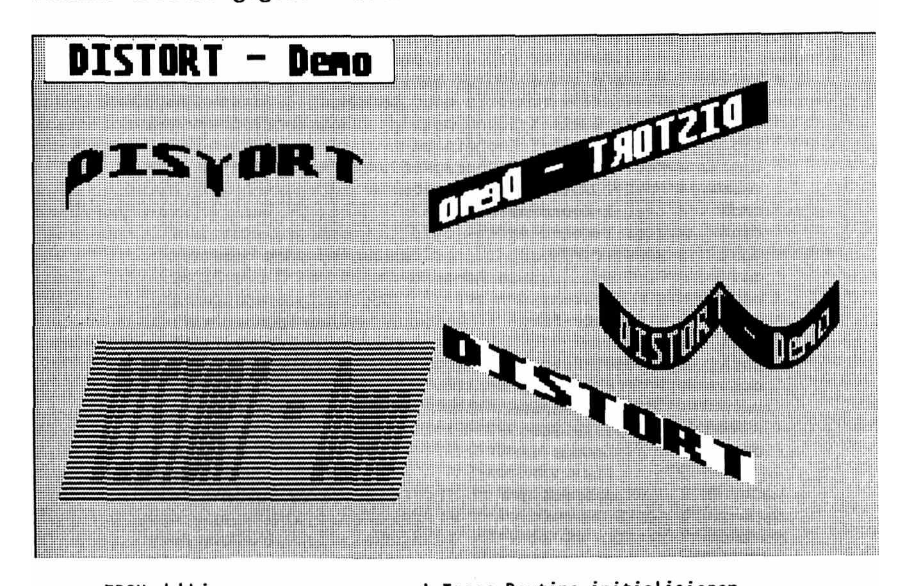
Memo Erwartet die Eingabe einer Adresse, ab welcher der aktuelle Speicherinhalt in Hex-Longs und ASCII-Zeichen ausgegeben wird.

Screen Erwartet die Eingabe einer Adresse, ab welcher 32000 Bytes gelesen und anschließend in den Bildschirmspeicher übertragen werden.

Ist die Zeilenanzeige geöffnet, kann generell durch erneutes Drücken der Anfangskombination <Control><Shift-links> das Programm normal fortgesetzt werden. Dies gilt auch im Einzelschritt-Verfahren.

Von der Routine werden bis auf zwei Kleinigkeiten keinerlei Veränderungen an vorhandenen Einstellungen vorgenommen. Die zwei Kleinigkeiten bestehen darin, daß bei jedem Durchlauf der PRINT-Zeilenüberlauf ein- (CHR$(27);"v" siehe PRINT) und ggfs. "revers"-PRINT ausgeschaltet wird (CHR$(27);"q" siehe PRINT).

Man soll sich ja nicht selbst auf die Schulter klopfen, aber diese Routine ist für meine Begriffe so gut gelungen, daß ich ihr liebevoll den Namen "Debbie" gegeben habe.




```
TRON debbie                         ! Trace-Routine initialisieren
xt%=2-SGN(XBIOS(4))                 ! X-Auflösungsteiler
yt%=MIN(2,3-XBIOS(4))               ! Y-Auflösungsteiler
DEFFILL ,2,4                        ! DEFFILL grau
PBOX 0,0,640/xt%,400/yt%
DEFFILL ,0,0                        ! DEFFILL weiß
PBOX 10/xt%,3/yt%,275/xt%,36/yt%
DEFTEXT ,1,,26/yt%                  ! DEFTEXT Fett-Mammut
TEXT 10/xt%,30/yt%,264/xt%," DISTORT - Demo "
@distort(9/xt%,3/yt%,265/xt%,34/yt%,300/xt%,120/yt%,...
                                    ...6/yt%,2,22,0,0,1,1,12)
@distort(28/xt%,7/yt%,145/xt%,30/yt%,310/xt%,220/yt%,...
                                    ...-15/yt%,2,22,0,0,0,2,3)
@distort(9/xt%,3/yt%,265/xt%,34/yt%,430/xt%,190/yt%,...
                                   ...35/yt%,2,1,0,1,0,0.7,13)
@distort(9/xt%,3/yt%,265/xt%,34/yt%,50/xt%,230/yt%,...
                                     ...250,2,2,1,0,0,4,11)
DIM b$(20)                          ! DIM Einzelbild-Puffer
PRINT AT(2,24);"Bitte Geduld"
GET 20/xt%,56/yt%,270/xt%,170/yt%,a$ ! Hintergrund sichern
FOR i%=5 TO 100 STEP 5              ! 20 mal
  @distort(28/xt%,7/yt%,145/xt%,30/yt%,(20+I%)/xt%,...
          ...110/yt%,-30+i%/2,1,1,0,1,0,2-(i%/50),7)
  GET 20/xt%,56/yt%,270/xt%,170/yt%,b$((i%-5)/5) ! GET Einzelbild
  PUT 20/xt%,56/yt%,a$              ! Hintergrund restauruieren
NEXT i%
DO                                  ! Kino-Schleife
  FOR i%=0 TO 20                    ! 20 mal vorwärts
    PUT 20/xt%,56/yt%+i%*2,b$(i%) ! Einzelbild zeichnen
```

```
    NEXT i%
    FOR i%=20 DOWNTO 0          ! 20 mal rückwärts
      PUT 20/xt%,56/yt%+i%*2,b$(i%) ! Einzelbild zeichnen
    NEXT i%
  LOOP
  PROCEDURE distort(dxsl%,dyso%,dxsr%,dysu%,dxd%,dyd%,...
                ...dwh%,dps,dsp,dvh%,dmd%,dsf%,dss,drm%)
    ' Verzerrt einen Bildausschnitt horizontal
    ' oder vertikal als Sinuswelle oder Gerade.
    ' dxsl%  = Linke Quell-X-Koordinate
    ' dyso%  = Obere Quell-Y-Koordinate
    ' dxsr%  = Rechte Quell-X-Koordinate
    ' dysu%  = Untere Quell-Y-Koordinate
    ' dxd%   = Ziel-X-Koordinate
    ' dyd%   = Ziel-Y-Koordinate
    ' dwh%   = Verzerrungsmaß           (beliebig, auch negativ)
    ' dps    = PI-Steps      (= Form: Realwert von 0 bis 2.00)
    ' dsp    = Sinus-Potenz               (beliebiger Realwert)
    ' dvh%   = Richtungs-Flag    (vertikal = 0/horizontal = 1)
    ' dmd%   = Modus                     (Gerade = 0/Kurve = 1)
    ' dsf%   = Spiegel-Flag                    (aus = 0/an = 1)
    ' dss    = Stretch-Faktor               (Realwert beliebig)
    '          Bei Dss-Werten > 2 wird nicht mehr ganzflächig
    '          dargestellt.
    ' drm%   = Grafikmodus                     (0 bis 15 siehe PUT)
    '
    LOCAL dbl%,dbr%,ds%,dl%,dvh,di,dlx% ! Lokale Variablen
    dbl%=dxsr%-dxsl%              ! Blockbreite
    dbr%=dysu%-dyso%              ! Blockhöhe
    xyd%=dyd%                     ! Vert. Zielkoordinate
    IF dvh%                       ! Horizontal?
      xyd%=dxd%                   ! Horiz. Zielkoordinate
      SWAP dbl%,dbr%              ! Breite und Höhe tauschen
    ENDIF
    DIM darr$(dbl%)               ! DIM Scan-Zeilen-Puffer
    FOR di=1 TO dbl%              ! Alle Scan-Zeilen
      IF dvh%                     ! Horizontal?
        GET dxsl%,dyso%+di,dxsr%,dyso%+di,darr$(di) ! Spalte lesen
      ELSE                        ! Vertikal!
        GET dxsl%+di,dyso%,dxsl%+di,dysu%,darr$(di) ! Zeile lesen
      ENDIF
    NEXT di
    FOR di=-PI TO PI STEP (PI/((dbl%)/(dps+1.0E-10)))
      ' Einmal rundum, alle Scan-Lines
      dl%=dwh%*ABS(SIN(di))^dsp+xyd% ! Zielpunkt für Scan-Line
      INC ds%                     ! Scan-Line-Zähler +1
      dlx%=ds%                    ! ...puffern
      IF dsf%                     ! Spiegel-Flag an?
        dlx%=(dbl%-ds%)+1         ! Dann rückwärts zählen
      ENDIF
      IF dmd%=0                   ! Als Gerade?
        dvh=dbr%/(dwh%+1.0E-10)   ! Neigungsverhältnis
        dl%=xyd%-(ds%/dvh)        ! Neuen Zielpunkt berechnen
      ENDIF
      IF dvh%                     ! Horizontal?
        PUT dl%,dyd%+ds%*dss,darr$(dlx%),drm%   ! 2 Spalten...
        PUT dl%,dyd%+ds%*dss+1,darr$(dlx%),drm% ! ...zeichnen
      ELSE                        ! vertikal!
        PUT dxd%+ds%*dss,dl%,darr$(dlx%),drm%   ! 2 Zeilen...
```

```
    PUT dxd%+ds%*dss+1,dl%,darr$(dlx%),drm% ! ...zeichnen
   ENDIF
   EXIT IF ds%=dbl%           ! Exit, wenn alle Scans durch
  NEXT di
  ERASE darr$()               ! Scan-Zeilen-Puffer löschen
RETURN
'
' **************** Die Debugging-Prozedur ****************** '
PROCEDURE debbie                ! Für alle drei Auflösungen !!
  ' vvvvvvv******FOLLOW******vvvvvvv
  '
  ' tron_var|=INSTR(UPPER$(TRACE$),UPPER$("VARIABLENNAME"))...
  '                                          ...*ABS(tron_nof!)
  '
  '   Soll eine Variable verfolgt werden, so ist sie bzw. die
  '   entsprechende Zuweisungsform (siehe oben unter <Shift><Delete>)
  '   in dieser INSTR()-Zeile als Suchbegriff in der hinteren
  '   UPPER$-Klammer als Text-String anzugeben ("VARIABLENNAME")
  '   und die Zeile zu aktivieren.
  '
  ' ^^^^^^^******FOLLOW******^^^^^^^
  IF (BIOS(11,-1) AND 15)=6 OR tron_new!=TRUE OR tron_var|
    ' <Control><Shift-links> gedrückt oder neue Posititon
    '  oder Variablen-Überwachung aktiv und Variable gefunden?
    WHILE (BIOS(11,-1) AND 15)=6 ! Solange <Control><Shift>
    WEND                         ! gedrückt wird -> WEND
    tron_xtl|=2-SGN(XBIOS(4))    ! X-Auflösungsteiler
    tron_ytl|=MIN(2,3-XBIOS(4))  ! Y-Auflösungsteiler
    tron_ylo%=XBIOS(2)+32000-1280! Adresse der untersten Zeile
    tron_clp|=16/tron_ytl|       ! Höhe einer PRINT-Zeile
    tron_byt|=80*tron_ytl|       ! Bytes pro Scan-Zeile
    tron_new!=FALSE              ! Positionsänderungs-Flag aus
    tron_ono!=tron_ono! XOR TRUE ! On/Off-Wechsel-Flag
    IF tron_ono!=TRUE            ! Ausgabe öffnen?
      tron_clr$=STRING$(tron_byt|,0) ! Scan-Zeilenlöscher
      tron_flg!=TRUE             ! Eingangs-Flag setzen
    ELSE                         ! Ausgabe schließen!
      BMOVE V:tron_pic$,tron_sta%,MAX(1,LEN(tron_pic$)) ! dann
    ENDIF                        ! ...Bildschirm restaurieren
  ENDIF
  tron_pos|=INSTR(UPPER$(TRACE$),UPPER$(tron_seq$)) ! Sequenz..
  '                              ! ...in TRACE$ enthalten?
  IF tron_ono!=TRUE OR tron_pos| OR tron_var| ! On/Off-Flag...
    ' ...gesetzt, Follow-Variable oder Sequenz gefunden?
    PAUSE tron_pau|              ! Ausführungspause
    IF tron_ono!=FALSE OR tron_flg!=TRUE ! Sequenz gefunden...
      '                          ! ...oder Öffnungs-Flag gesetzt?
      tron_flg!=FALSE            ! Öffnungs-Flag löschen
      tron_ln2|=tron_ln1|        ! Neue Zeilenposition
      tron_ln3&=tron_ln2|*1280   ! Fenstergröße in Bytes
      tron_sta%=tron_ylo%-tron_ln3&-tron_scn|*tron_byt|
      ' Startadresse des Anzeige-Fensters
      tron_pic$=SPACE$(tron_ln3&+1280+tron_scn|*tron_byt|)
      ' Hintergrund-Puffer vorbereiten
      BMOVE tron_sta%,V:tron_pic$,MAX(1,LEN(tron_pic$))
      ' Ausgabe-Hintergrund in Puffer übertragen
      tron_ono!=TRUE             ! On/Off-Flag setzen
    ENDIF
    BMOVE V:tron_clr$,tron_ylo%+1280-tron_ln3&-tron_byt|,...
```

```
                                         ...MAX(1,tron_byt|)
' Unterste Scan-Zeile der Ausgabe löschen (Kosmetik)
tron_sta%=tron_ylo%-tron_ln3&-tron_scn|*tron_byt|
' Neue Startadresse des Ausgabebereichs berechnen
FOR i%=0 TO 15/tron_ytl|       ! 16 bzw. 8 mal
  BMOVE tron_sta%+tron_byt|,tron_sta%,MAX(1,(1280-...
                   ...tron_byt|)+tron_scn|*tron_byt|)
  ' Ausgabe-Fenster je um eine Scan-Zeile aufwärts
NEXT i%
PRINT CHR$(27);"w";            ! Zeilenüberlauf aus
tron_csc|=CRSCOL               ! Akt. Cursor-Spalte speichern
tron_csl|=CRSLIN               ! Akt. Cursor-Zeile speichern
PRINT AT(1,25-tron_ln2|);LEFT$(TRACE$,80/tron_xtl|); !TRACE$
PRINT CHR$(27);"K";            ! Zeilenrest löschen
IF tron_pos|                   ! Such-Sequenz gefunden?
  PRINT CHR$(27);"p";          ! PRINT revers
  FOR tron_cnt|=1 TO MIN(80/tron_xtl|,LEN(TRACE$))
    ' Alle Zeichen von TRACE$ einzeln...
    PRINT AT(tron_cnt|,25-tron_ln2|);MID$(TRACE$,tron_cnt|,1);
    ' Schwarz schreiben (optisches Signal für "gefunden")
  NEXT tron_cnt|
  PRINT CHR$(27);"q";          ! PRINT schwarz auf weiß
  PRINT AT(1,25-tron_ln2|);LEFT$(TRACE$,80/tron_xtl|);
  ' TRACE$ wieder normal ausgeben
  IF tron_pos|<(80/tron_xtl|)! Gefundene Sequenz im...
    '                          ! ...Bildschirmbereich?
    PRINT CHR$(27);"p";        ! PRINT revers
    PRINT AT(tron_pos|,25-tron_ln2|);LEFT$(tron_seq$,...
                        ...(80/tron_xtl|-tron_pos|+1));
    ' Gefundene Sequenz schwarz markieren
    PRINT CHR$(27);"q";        ! PRINT schwarz auf weiß
  ENDIF
ENDIF
IF (BIOS(11,-1) AND 15)=2      ! Linke <Shift>-Taste gedrückt?
  KEYTEST tron_key%            ! Tastatur abfragen
  LPOKE XBIOS(14,1)+6,0        ! Tastatur-Puffer löschen
  SELECT tron_key%             ! <Taste> selektieren
  CASE &H2480038               ! <Pfeil-hoch>-Taste gedrückt?
    IF tron_scn|<96/tron_ytl| AND tron_ln2|<19
      ' Unterste Zeile unterhalb der 6. Cursor-Zeile und
      ' Ausgabehöhe kleiner als 6 Cursor-Zeilen?
      ADD tron_scn|,tron_clp| ! Fensterhöhe + Cursor-Höhe
      tron_buf$=SPACE$(tron_clp|*tron_byt|) ! Zeilen-Puffer
      tron_sta%=tron_ylo%-tron_ln3&-tron_scn|*tron_byt|
      ' Neue Startadresse des Ausgabebereichs berechnen
      BMOVE tron_sta%,V:tron_buf$,MAX(1,tron_clp|*tron_byt|)
      ' Zeilenhintergrund in den Puffer übertragen
      tron_pic$=tron_buf$+tron_pic$ ! Neue Zeile in den...
    ENDIF                      ! ...Haupt-Puffer einbinden
  CASE &H2500032               ! <Pfeil-runter>-Taste gedrückt?
    IF tron_scn|>0             ! Zusatzzeilen bereits offen?
      SUB tron_scn|,tron_clp|! Fensterhöhe - Cursor-Höhe
      tron_sta%=tron_ylo%-tron_ln3&-tron_scn|*tron_byt|
      ' Neue Startadresse des Ausgabebereichs berechnen
      BMOVE V:tron_pic$,tron_sta%-tron_clp|*tron_byt|,...
                          ...MAX(1,tron_clp|*tron_byt|)
      ' Zeilenhintergrund restaurieren
      tron_pic$=RIGHT$(tron_pic$,LEN(tron_pic$)-...
                          ...tron_clp|*tron_byt|)
```

```
    ' Zeile aus dem Haupt-Puffer löschen
  ENDIF
CASE &H24B0034              ! <Pfeil-links>-Taste gedrückt?
  IF tron_scn¦=0            ! Keine Zusatzzeilen offen?
    tron_ofs¦=24            ! Dann Ausgabezeile bis...
    '                       ! ...ganz oben möglich
  ELSE                      ! Zusatzzeile(n) offen!
    tron_ofs¦=18            ! Dann nur bis zur 6. Zeile
  ENDIF
  IF tron_ln2¦<tron_ofs¦   ! Ausgabezeile noch nicht oben?
    tron_ln1¦=MIN(tron_ln2¦+1,tron_ofs¦) ! Neue Zeilenposition
    tron_ono!=FALSE         ! On/Off-Flag ausschalten
    tron_new!=TRUE          ! Positionsänderungs-Flag setzen
    PAUSE 2                 ! Kleine Pause, damit der Tasten-Code
    ' nicht ans Hauptprogramm weitergeht (nicht ganz vermeidbar).
    ENDIF
  CASE &H24D0036              ! <Pfeil-rechts>-Taste gedrückt?
    IF tron_ln2¦>0            ! Ausgabezeile noch nicht unten?
      tron_ln1¦=MAX(tron_ln2¦-1,0) ! neue Zeilenposition
      tron_ono!=FALSE         ! On/Off-Flag ausschalten
      tron_new!=TRUE          ! Positionsänderungs-Flag setzen
      PAUSE 2                 ! Kleine Pause
    ENDIF
  CASE &H2470037              ! <ClrHome>-Taste gedrückt?
    tron_pau¦=(tron_pau¦+4) MOD 44 ! PAUSE-Wert setzen
    PRINT AT(70-(40*(tron_xtl¦-1)),25-tron_ln2¦);"PAUSE:";...
            ...tron_pau¦;   ! Pausenwert anzeigen
  CASE &H2610000              ! <Undo>-Taste gedrückt?
    tron_stp!=TRUE            ! Einzelschrittmodus an
  CASE &H253007F              ! <Delete> gedrückt?
    tron_nof!=tron_nof! XOR TRUE ! Follow an/aus
  ENDSELECT
ENDIF
IF tron_stp!=TRUE OR tron_pos¦ OR tron_var¦ !Einzelschritt-
  ' Modus an, bzw. Follow-Variable oder Sequenz gefunden?
  tron_stp!=FALSE             ! Einzelschritt-Flag aus
  ' vvvvvvv******FOLLOW******vvvvvvv
  '
  IF tron_var¦                ! Follow-Variable gefunden?
    ' ------------Verfolge numerische Variable-------------
    '   Soll eine numerische Variable verfolgt werden, so ist
    '   sie hier hinter tron_var= anzugeben und die beiden
    '   Programmzeilen zu aktivieren.
    ' tron_var=dbl%           ! Beliebiger Typ (%!&¦)
    '   z.B.:  tron_var=Var&
    ' PRINT AT(80/tron_xtl¦-LEN(STR$(tron_var))-7,...
            ...25-tron_ln2¦);CHR$(27);"K";" ¦VAR:";tron_var;
    ' -----------Verfolge alphanumerische Variable----------
    '   Soll eine String-Variable verfolgt werden, so ist sie
    '   hier hinter tron_vr2$= anzugeben und die folgenden
    '   vier Programmzeilen zu aktivieren.
    ' tron_var$=a$            ! z.B.:  tron_var$=Var$
    ' PRINT AT(1,25-tron_ln2¦);CHR$(27);"K";"VAR$:";...
                      ...LEFT$(tron_var$,80/tron_xtl¦-5);
    ' KEYGET tron_key%        ! auf <Taste> warten
    ' PRINT AT(1,25-tron_ln2¦);LEFT$(TRACE$,80/tron_xtl¦);
  ENDIF                       ! TRACE$ nochmal ausgeben
  '
```

```
' ^^^^^^^******FOLLOW******^^^^^^^
REPEAT                          ! Einzelschritt-Schleife
  KEYTEST tron_key%             ! Tastatur abfragen
  SELECT tron_key%              ! <Taste> selektieren
  CASE &H2620000                ! <Shift><Help> gedrückt?
    SGET tron_scr$              ! Bildschirm sichern
    STOP                        ! In Direktmodus wechseln
    ' Nach Fortsetzung durch CONT im Direktmodus:
    TRON debbie                 ! TRON neu initialisieren
    SPUT tron_scr$              ! Bildschirm restaurieren
    CLR tron_scr$               ! Puffer löschen
    GOTO tron_lab               ! Zum Ausgangs-Label
  CASE &H2520030                ! <Shift><Insert> gedrückt?
    SGET tron_scr$              ! Bildschirm sichern
    PRINT AT(1,25-tron_ln2|);CHR$(27);"K";CHR$(27);"w";
    ' Cursor positionieren, Zeile löschen und Überlauf aus
    PRINT "<1>=Sequenz|<2>=DUMP|<3>=Memo|<4>Screen|"; ! Menü
    KEYGET tron_mod|            ! Auf <Taste> warten
    SELECT tron_mod|            ! <Taste> selektieren
    CASE "1"                    ! "1" gedrückt?
      PRINT AT(1,25-tron_ln2|);CHR$(27);"K";CHR$(27);"w";
      ' Cursor positionieren, Zeile löschen und Überlauf aus
      INPUT "Sequenz => ",tron_seq$  ! Zeilensequenz eingeben
    CASE "2"                    ! "2" gedrückt?
      PRINT AT(1,25-tron_ln2|);CHR$(27);"w";
      ' Cursor positionieren und PRINT "Überlauf aus"
      PRINT "DUMP => ";
      PRINT CHR$(27);"K";  ! Zeile löschen
      FORM INPUT 2,tron_dmp$! DUMP-Parameter eingeben
      CLS                       ! Bildschirm löschen
      DUMP tron_dmp$            ! DUMP ausführen
      PRINT CHR$(10);CHR$(13);" Weiter mit <Taste>";
      KEYGET tron_key%          ! Auf <Taste> warten
    CASE "3"
      PRINT AT(1,25-tron_ln2|);! Cursor positionieren
      PRINT "Start-Adresse (HEX='$....') => ";
      PRINT CHR$(27);"K";  ! Zeilenrest löschen
      FORM INPUT 7,tron_adr$ ! Startadresse eingeben und...
      tron_adr%=INT(VAL(tron_adr$)/2)*2 ! Auf "gerade" trimmen
      CLS                       ! Bildschirm löschen
      PRINT "Start-Adresse :"'tron_adr%;CHR$(10)
      FOR tron_cnt|=0 TO 69-(42*(tron_xtl|-1)) ! Nach Auflösung
      IF tron_cnt| MOD 10=0 ! Alle zehn Longwords...
        PRINT "Offset : +";tron_cnt|*4 ! ...neue Zeile
      ENDIF
      tron_lon%=LPEEK(tron_adr%+tron_cnt|*4) ! LPEEKen
      PRINT 'RIGHT$("00000000"+HEX$(tron_lon%),8);'
      ' HEX-Wert formatieren und ausgeben
      OUT 5,BYTE{V:tron_lon%},BYTE{V:tron_lon%+1},...
          ...BYTE{V:tron_lon%+2},BYTE{V:tron_lon%+3}
      ' Einzel-Bytes als Textzeichen ausgeben
      PRINT '"|";SPACE$(4*(tron_xtl|-1));
    NEXT tron_cnt|
    PRINT CHR$(10);CHR$(13);" Weiter mit <Taste>";
    KEYGET tron_key%          ! Auf <Taste> warten
  CASE "4"                    ! "4" gedrückt?
    PRINT AT(1,25-tron_ln2|);! Cursor positionieren
    PRINT "Start-Adresse (HEX='$....') => ";
    PRINT CHR$(27);"K";  ! Zeile löschen
```

```
            FORM INPUT 7,tron_adr$ ! Startadresse eingeben
            tron_adr%=MAX(2048,INT(VAL(tron_adr$)/2)*2) ! "trimmen"
            ' minimale Adresse = 2048 (nur im User-Bereich)
            BMOVE tron_adr%,XBIOS(2),32000 ! 32000 Bytes in den
            ' Bildschirmspeicher übertragen
            PRINT AT(1,1);" Weiter mit <Taste>";
            KEYGET tron_key%     ! Auf <Taste> warten
          ENDSELECT
          SPUT tron_scr$         ! Bildschirm restaurieren
          CLR tron_scr$          ! Puffer löschen
        ENDSELECT
        IF tron_key%=&H610000    ! <Undo> gedrückt?
          tron_stp!=TRUE         ! Einzelschritt-Flag wieder an
          PAUSE 10               ! Kleine Pause
        ENDIF
        EXIT IF tron_key%=&H2610000    ! Exit, wenn <Shift><Undo>
        EXIT IF (BIOS(11,-1) AND 15)=6 ! Exit, wenn
        ' <Control><Shift-links> gedrückt
      UNTIL tron_stp!=TRUE       ! Exit, wenn neues Flag gesetzt
      LPOKE XBIOS(14,1)+6,0      ! Tastatur-Puffer löschen
    ENDIF
    PRINT AT(tron_csc|,tron_csl|);CHR$(27);"v"; ! Cursor auf...
    '           ...alte Position setzen und Zeilenüberlauf "an"
    IF tron_ono!=FALSE           ! On/Off-Flag aus?
      BMOVE V:tron_pic$,tron_sta%,MAX(1,LEN(tron_pic$)) ! dann...
      '                          ! ...Bildschirm restaurieren
    ENDIF
  ENDIF
  tron_lab:                      ! Ausgangs-Label für CONT-Sprung
RETURN                           ! .........und zurück
```

Und nun die versprochene Variablenliste:

tron_adr$ String-Eingabepuffer für Adresse bei Memo, Screen.

tron_adr% Startadresse für Memo und Screen.

tron_buf$ Zwischenpuffer für eine Hintergrund-Zeile.

tron_byt| Enthält die Anzahl der Bytes pro Scan-Zeile (Hires=80/Midres=160/Lowres=160).

tron_clp| Enthält die Höhe einer Cursor-Zeile in Pixel (Hires=16/Midres=8/Lowres=8).

tron_clr$ Enthält eine Scan-Zeile mit Null-Bytes zum Löschen der untersten Ausgabe-Scan-Zeile (Bildschirmkosmetik).

tron_cnt| Schleifenzähler für die verwendeten Zähl-Schleifen.

tron_csc| Gemerkte Cursor-Spalte vor der Anzeige.

tron_csl| Gemerkte Cursor-Zeile vor der Anzeige.

tron_dmp$ String-Eingabepuffer für DUMP-Parameterzeile.

tron_flg! Flag zum Öffnen der Ausgabe (quasi der Schlüssel).

tron_key% Puffer für gedrückte Tastatur-<Taste>.

tron_ln1| Bei Positionsänderung ist dies die neue Cursor-Zeile.

tron_ln2| Enthält die aktuelle Ausgabe-Cursor-Zeile.

tron_ln3& Enthält die Größe des Ausgabe-Fensters in Bytes (Anzahl der Scan-Lines * Bytes pro Scan-Line).

tron_lon% Jeweils gelesenes Longword bei Memo.

tron_mod| Die beim Einzelschritt-Menü gedrückte <Taste>.

tron_new! Flag, das aussagt, daß bei Positionsänderung des Ausgabebereichs beim nächsten Durchlauf die Ausgabe automatisch wieder geöffnet wird.

tron_nof! Wechsel-Flag, das darüber entscheidet, ob die angegebene Follow-Variable überwacht werden soll oder nicht (siehe oben unter <Shift>-<Delete>).
Dieses Flag ist bei Programmstart ausgeschaltet, da ggfs. bei Überwachung von Feld-Variablen erst die zugehörige DIM-Anweisung übersprungen werden muß, um eine "Feld nicht dimensioniert"-Fehlermeldung zu vermeiden.

tron_ofs| 25 minus tron_ofs| ergibt die Cursor-Zeile, bis zu welcher die unterste Ausgabezeile verschoben werden kann.

tron_ono! Wechsel-Flag, daß bei jedem Druck auf <Control> und <Shift-links> an- und ausgeschaltet wird. Es entscheidet darüber, ob das Fenster geöffnet wird.

tron_pau| PAUSE-Wert für verzögerte Programmausführung.

tron_pic$ Enthält den aus Einzel-Cursor-Zeilen zusammengesetzten Fensterhintergrund.

tron_pos| Ggfs. Position der gesuchten Zeilensequenz.

tron_scn| Enthält die Anzahl der Scan-Lines, die die zusätzlichen Ausgabezeilen benötigen.

tron_scr$ Puffer für die Fullscreen bei DUMP, Memo, Screen.

tron_seq$ Enthält Zeilensequenz, auf die gewartet werden soll.

tron_sta% Startadresse des Ausgabefensters.

tron_stp! Wechsel-Flag, das darüber entscheidet, ob im Einzelschrittmodus verharrt werden soll.

tron_var| Enthält ggfs. die Position des Follow-Variablennamens in der aktuelle TRACE$-Zeile.

tron_var Numerische Real-Puffervariable für numerische Follow-Variablen beliebigen Typs.

tron_var$ String-Puffervariable für alphanumerische Follow-Variablen.

tron_xtl| X-Auflösungsteiler (Hires = 1/Midres = 1/Lowres = 2).

tron_ylo% Startadresse der untersten Ausgabezeile.

tron_ytl| Y-Auflösungsteiler (Hires = 1/Midres = 2/Lowres = 2).

Übrigens ist keine dieser Variablen lokal deklariert, da ich meine, daß bei der Wahl der Namen (alle beginnen mit tron_) die Gefahr einer Kollision mit den Hauptprogramm-Variablen nicht gegeben ist.

12.9 Diverses

Version 3.0

$ — Textbereich für V3.0-Compiler deklarieren

$ Text

Deklariert beliebige Programmzeilen als Textspeicher zur Weiterverarbeitung durch den V3.0-Compiler. Im Interpreter-Betrieb wird eine solche Zeile wie REM bzw. ' behandelt. Nähere Informationen sind erst mit Erscheinen des V3.0-Compilers zu erwarten.

DEFLIST { DEFLIS } Listing-Format festlegen

DEFLIST Format

Format:

0	Befehlsnamen werden groß, Variablennamen klein geschrieben (PRINT var$). In V3.0 wird bei global deklarierten Variablennamen das Postfix vernachlässigt (Postfix siehe DEFxxx-Befehle DEFBYT etc.).
1	Anfangsbuchstaben von Befehls- und Variablennamen werden groß geschrieben (Print Var$). In V3.0 wird bei global deklarierten Variablennamen das Postfix vernachlässigt.

In V3.0:

2	Wie 0, jedoch wird bei Variablennamen ggfs. das Postfix mit angezeigt.
3	Wie 1, jedoch wird bei Variablennamen ggfs. das Postfix mit angezeigt.

DEFLIST kann nur im Interpreter-Direktmodus verwendet werden.

DEFNUM { DEFN } Rundung von Ziffern-Ausgaben

DEFNUM Stelle

Stelle gibt die Ziffernstelle an (in V2.xx: 3 - 11/in V3.0: 3 - 13), auf die alle folgenden, - durch PRINT etc. - auszugebenden Werte gerundet werden sollen. Diese Einstellung wird erst durch den nächsten DEFNUM-Befehl verändert. Durch Angabe der maximalen Stellenanzahl (11 bzw. 13) wird die normale Werteausgabe wieder angeschaltet.

Bei Realzahlen wird der Vorkomma-Anteil, der ggfs. hinter Stelle liegt, als Nullen ausgegeben. Liegt Stelle im Nachkommabereich, werden alle dahinterliegenden Nachkommastellen ignoriert bzw. die nächste Stelle hinter Stelle wird zur Rundung verwendet.

Die Rundung erfolgt mathematisch exakt (Int(A+0.5)). In diesem Format ausgegebene Variableninhalte bleiben von DEFLIST unberührt. Die interne Rechengenaugkeit wird hierdurch nicht beeinträchtigt.

Beispiel:

```
A=Random(1000000)+Rnd
Print "Originalwert: ";A;Chr$(10)
```

```
For I%=3 To 11
  Defnum I%
  Print A,"DEFNUM ";I%
Next I%
```

FALSE — Unwahr-Konstante

Var=FALSE

Reservierte Variable. Enthält konstant den Un-Wert 0.

LET { LE } — Daten zuweisen

LET Var=Wert
LET Var$=Text

Es wird Wert bzw. Text der angegebenen Variablen zugewiesen.

Die Benutzung von LET birgt bei der Version 2.0 den Vorteil, daß die Variable Var mit einem beliebigen Namen (auch Befehlsnamen, außer Namen reservierter Variablen wie PI, ERR, FATAL, TIMER etc.) benannt werden kann.

Ab V2.02 ist LET überflüssig, da es dort bei Variablennamen keine Einschränkungen (außer PI, ERR, FATAL, TIMER etc.) mehr gibt. Hier hat LET nur noch den Zweck der Kompatibilität zu den BASIC-Dialekten (z.B. ST-BASIC), deren Programme man durch Merge in den GFA-Arbeitsspeicher laden kann.

Version 3.0

MODE { MOD } — Zahlen/Datum -> Europa/USA wählen

MODE Modus

In der Grundeinstellung werden in GFA-BASIC durch PRINT USING Werte im Europa-Format ausgegeben. D.h. ggfs, daß als Tausendertrennung ein Komma und als Dezimaltrennung ein Punkt verwendet wird. In den USA gelten die umgekehrten Konventionen. Ähnliches gilt für die Darstellung des Datums (siehe SETTIME).

Modus ist als 2-Bit-Vektor anzusehen, dessen Bit 0 die Darstellung des Datums durch DATE$ und FILES bzw. über das Eingabeformat des Datums bei SETTIME und DATE$= bestimmt. Bit 1 entscheidet über

die zu verwendende Art der Werte-Darstellungsart bei PRINT USING-Ausgaben bzw. bei Verwendung von STR$().

Modus	USING	DATE$
&X00	#,###.##	25.05.1988
&X01	#,###.##	05/25/1988
&X10	#.###,##	25.05.1988
&X11	#.###,##	05/25/1988

OPTION { OPT } Compiler-Steuerung

OPTION "Anweisung"

- Gilt vorerst nur für die Versionen V2.xx -

Dem GFA-BASIC-V2.xx-Compiler können Arbeitsanweisungen übermittelt werden, die darüber entscheiden, ob bestimmte Compileroptionen in das jeweilige GFA BASIC-Compilat eingebaut werden sollen oder nicht.

Anweisung: Bedeutung:

Anweisung	Bedeutung
"U0"	Die Break-Funktion <Control/Shift/Alternate> wird nicht in das Compilat eingebaut. Eine Programmverzweigung durch ON BREAK GOSUB ist **nicht** möglich.
"U1"	An der Programmstelle, wo "U1" verfügt wird, wird nur **einmal** überprüft, ob die Break-Tasten gedrückt wurden.
"U2"	Vor jedem Schleifen-Wendepunkt (LOOP,UNTIL,WEND,NEXT), sowie vor jedem GOTO-Befehl wird eine Abfrage eingebaut.
"U3"	Es werden **nach** jedem Befehl einmal die Breaktasten abgefragt. Das Compilat reagiert dann wie der Interpreter, ist jedoch wesentlich langsamer als bei "U0" bzw. "U1".
"T-"	Ein evtl. auftretender Integer-Überlauf (z.B. A%=2^32) wird **nicht** kontrolliert.
"T+"	Es wird eine Routine in das Compilat eingefügt, die den Integer-Überlauf kontrolliert und ggfs. eine Fehlermeldung produziert.
"E-"	Bei Fehlermeldungen wird lediglich die Fehlernummer (siehe ERR), sowie der Hinweis "PC>$00xxxxxx" ausgegeben.
"E+"	Es werden im Compilat ausführliche Fehlermeldungen ausgegeben.
"B-"	Bei Bomben-Fehlern (ERR > 100) erolgt der schon fast vergessene Parade-Absturz. Der ON ERROR GOSUB-Befehl bleibt wirkungslos.
"B+"	Es werden Routinen in das Compilat eingebaut, die auch Bomben-Errors abfangen und ggfs. eine Fehlermeldung ausgeben bzw. zu einer angegebenen ON ERROR GOSUB-Routine verzweigen.

Bei allen "+"-Optionen ist mit teilweise erheblichen Geschwindigkeitsverlusten und umfangreicheren Compilaten zu rechnen.

Widersprechen sich die Options-Verfügungen, registriert der Compiler nur die letzte Verfügung. Wurde z.B. zuerst "U3" verfügt und an spä-

terer Programmstelle "U0", wird die Break-Routine nicht eingebunden, da nur die letzte Verfügung ("U0") Gültigkeit behält (vgl. 24.1 "Der V2.xx-Compiler").

TRUE Wahr-Konstante

Var=TRUE

TRUE ist eine reservierte Variable, die konstant den Wert -1 enthält.

Bei verschiedenen Funktionen (z.B. EXIST) wird ein Wahrheitswert zurückgegeben. Zur besseren Übersichtlichkeit kann bei Bedingungsabfragen dieser Art, bei Bit-Flag-Verwaltung oder Zuweisungen etc., statt des Wertes -1 die Konstante TRUE verwendet werden (siehe auch FALSE).

Beispiel 1:

```
Fileselect "\*.*","",F$      ! Datei wählen
If Exist(F$)=True            ! Datei auf Disk?
  ... weiteres
  ... Programml
```

Beispiel 2:

```
Do                              ! Endlos-Schleife
  If Mousek=1                   ! Linke Maustaste gedrückt?
    Bitflag!=Bitflag! Xor True ! Flag bei jedem Mausklick
    '                           ! Abwechselnd an- und ausschalten
    Onoff%=Abs(Bitflag!=True)  ! Ergibt: An = 1/Aus = 0
    Print At(1,1);Right$("an ",3*Onoff%)
    Print At(1,1);Right$("aus",3*Abs(Onoff%=False))
    Pause 6                     ! Kleine Klickpause
  Endif
  Circle Mousex*Onoff%,Mousey*Onoff%,10*Onoff%
Loop
```

Vielleicht können Sie im obigen Beispiel auf Anhieb nicht viel mit den beiden PRINT-Zeilen anfangen. Dadurch, daß ich eine Wahrheitsabfrage in die Zeilen mit einbaue, erspare ich mir eine entsprechende IF-Abfrage. Die Variable Onoff% enthält je nachdem, ob das Bitflag! -1 oder 0 ist, den Wert 1 oder 0. Mit der RIGHT$-Konstruktion lasse ich nur dann die Ausgabe des Strings zu, wenn der Wert 3 (Länge des Strings) mit 1 multipliziert wird. Steht in Onoff% der Wert Null, ist die zu ermittelnde RIGHT$-String-Länge ebenfalls Null und es wird kein String ausgegeben.

Im Normalverfahren würde die Konstruktion so aussehen:

```
If Bitflag!=True
  Print At(1,1);"an "
Else
  Print At(1,1);"aus"
Endif
```

SWAP { SW } Variablen/Felder/Pointer tauschen

SWAP Var1,Var2
SWAP Element(x),Element(y)
SWAP Feld1(),Feld2()
SWAP *Pointer,Feld()

Dies ist der SWAP-Befehl, in V3.0 gibt es SWAP() auch als Funktion.

SWAP tauscht die Inhalte zweier gleichartiger Variablen, Felder oder Feldelemente (numerische oder alphanumerische), bzw. einen Zeiger auf einen Feld-Descriptor (siehe *) mit dem Descriptor des angegebenen Feldes (Feldübergabe an Prozeduren). Bei Feldern wird auch die Dimensionierung vertauscht.

Beispiel 1:

```
A%=100
B%=1000
Print "A%, B% vor SWAP:";A%'B%
Swap A%,B%
Print "A%, B% nach SWAP:";A%'B%
```

Beispiel 2:

Ab der Version V3.0 ist es bei Pointer/Feld-SWAP nicht mehr nötig, das angegebene Feld vorher zu dimensionieren.

```
Dim A%(10)
@Routine(*A%())
Procedure Routine(X%)
  ' Dim Dummy%(1)      ! <--- Nur in V2.xx nötig!
  Swap *X%,Dummy%()
  .
  ... Ab hier ist das vorherige Feld A%() unter
  ... dem Namen Dummy%() ansprechbar. Feld A%()
  ... ist leer.
  .
  ... Routinentext
  .
  Swap *X%,Dummy%()
  .
  ... Ab hier ist Feld Dummy%() wieder Feld A%()
  ... und Feld Dummy%() ist wieder leer.
  .
```

```
  ' Erase Dummy%()       ! <--- Nur in V2.xx nötig!
Return
```

Diese indirekte Übergabe von Feldern an Prozeduren hat den Vorteil, daß das durch SWAP in die Prozedur "hineingetauschte" Feld nicht unter seinem eigentlichen (globalen) Namen angesprochen werden muß, sondern innerhalb der Prozedur unter einem lokalen Namen angesprochen und bearbeitet werden kann.

Genaugenommen ist diese Kontruktion jedoch in V3.0 überflüssig, da hier Felder genauso wie Variablen durch VAR direkt an die Prozedur übergeben werden können:

```
DIM feld%(1)
xyz(feld%())
PRINT Feld%(1)
PROCEDURE Xyz(VAR Dummyfeld%())
  Dummyfeld%(1)=100
RETURN
```

Weitere Beispiele hierzu finden Sie unter GET (Prozedur Menue), BMOVE (Prozedur Screen), SSORT (Prozedur Sort), sowie in 9.5.1 "Organisation eines PUT-Strings" in der Prozedur Gplane.

VOID {VO} — Dummy-Zuweisung

In V3.0: { ~ }

VOID Funktion

VOID ist ein Ersatz für eine sogenannte Dummy-Zuweisung, jedoch - vor allem in Compilaten - ergeblich schneller. Es wird eine Funktion aufgerufen, ohne dieser eine Rückgabe-Variable zur Verfügung stellen zu müssen. In vielen Fällen ist die Zuweisung des Funktionsergebnisses an eine Aufnahme-Variable bei Funktionsaufrufen unnötig, da das Ergebnis nicht von Interesse ist. Um Speicherplatz (und Zeit) zu sparen, können diese Funktionen mit VOID aufgerufen werden.

Beispiel:

```
Statt:  A=Inp(2)          ! Auf Taste warten
        -->  Void Inp(2)
Statt:  A=Fre(0)          ! Garbage-Collection
        -->  Void Fre(0)
Statt:  A=Gemdos(&HE,1)   ! Disk B aufrufen
        -->  Void Gemdos(&HE,1)
```

In Version V3.0 kann ~ verwendet werden (~GRAF_GROWBOX).

13. Interaktionen (Programm/Benutzer)

ALERT {A} — Alert-Box erstellen

ALERT,Icon%,B_oxtext$,Def_Button%,Buttontext$,Backvar%

Diese GEM-Funktion erlaubt es, mit dem Anwender unkompliziert zu kommunizieren. Prinzipiell lassen sich damit sogar umfangreiche Menüs verwalten. Da die ALERT-Box (engl.: wachsam,alarmbereit) aber nur maximal drei Auswahlmöglichkeiten zur Verfügung stellt, ist sie eher für kleinere Abfragen oder Hinweistexte geeignet.

Icon%

0 = Kein Symbol	2 = Fragezeichen
1 = Ausrufungszeichen	3 = STOP-Schild

B_oxtext$

Hier wird der eigentliche Text (Mitteilung/Frage) an die Funktion übergeben. Das Pipe-Zeichen | gilt darin als Trennungszeichen zwischen den einzelnen Zeilen. Es können insgesamt 4 Zeilen zu maximal je 30 Zeichen dargestellt werden. Der Text kann direkt in den Befehl geschrieben, als String-Ausdruck oder auch als String-Variable übergeben werden.

Def_Button%

Es wird die Nummer (1,2,3) des Buttons übergeben, welcher außer durch Mausklick auch durch die <Return>-Taste (default) bestätigt werden kann. Dieser Button wird in der Box stark umrandet gezeichnet (0 = Kein Default-Button).

Buttontext$

Hierdurch erfolgt die Beschriftung der Buttons. Jeder Button kann mit jeweils maximal 8 Zeichen beschriftet werden. Auch hier gilt das |-Zeichen als Trennstrich zwischen den einzelnen Button-Texten.

Backvar%

Numerische Variable, in welcher der Befehl die Nummer des bestätigten Button (1,2,3) zurückgibt.

Beispiel:

```
B_oxtext$="Klicken Sie bitte eines¦der unteren Kästchen an!"
B_oxtext$=B_oxtext$+"¦Ich sage Ihnen dann, welches¦es war!"
Buttontext$="XX 1 XX¦XX 2 XX¦XX 3 XX"
Alert 1,B_oxtext$,0,Buttontext$,Backvar%
B_oxtext$="Es war :          ¦ ¦das Kästchen "+Str$(Backvar%)
Buttontext$=" OKAY ¦ OKAY ¦ OKAY "
Alert 1,B_oxtext$,2,Buttontext$,Backvar%
```



Diese Form der Alert-Box ist aus Gründen der Sicherheit nicht besonders variabel. Vom Interpreter werden sämtliche Eingaben auf ihre Zulässigkeit überprüft. Werden z.B. mehr als 30 Zeichen Boxtext in einer Zeile angegeben, wird die Zeile vom Interpreter nach dem 30sten Zeichen abgeschnitten.

Wer die eigentliche GEM-Alert-Box kennt, der wird diese Sicherheitsmaßnahmen verstehen. Wer trotzdem etwas mehr Freiheit in der Gestaltung einer Alert-Box genießen möchte, hat in V3.0 die Möglichkeit, die entsprechende GEM-Routine als Funktion FORM_ALERT() aufzurufen.

Wer weiter in V2.xx programmiert und diese "echte" Alert-Box verwenden will, kann sich der folgenden Prozedur Form_alert bedienen, die (fast) dasselbe macht wie die V3.0-Funktion FORM_ALERT.

Während in V3.0 versucht wird, die Eingaben bis zu einem gewissen Grad auf Ihre Zulässigkeit zu überwachen, wird bei dieser Prozedur in V2.xx von Systemseite aus **keine** Rücksicht auf falsche Eingaben oder zu große Textlängen genommen. Ist der Text bei dieser "echten" GEM-Alert-Box zu lang, macht das System schlicht und ergreifend eine Bauchlandung, vor der Sie auch der Interpreter mit seinem Absturzverhinderer nicht mehr retten kann.

Von Digital Research vorgegeben ist hier ebenfalls eine Zeilenlänge von 30 Zeichen bei insgesamt 5 Zeilen und eine Buttontextlänge von 10 Zeichen je Button. Das sind immerhin 30 Zeichen mehr an reinem Boxtext. Allerdings ist in V2.xx auch folgende Variante möglich, die jedoch in V3.0 gnadenlos abstürzt:

```
@Form_alert(0,String$(8,"BOXTEXT"),1,String$(12,"BUTTON"),*Back%)
Procedure Form_alert(Symbol%,Bxtxt$,Button%,Bttxt$,Done%)
  Local Alstring$
```

```
    Alstring$="["+Str$(Symbol%)+"]["+Bxtxt$+"]["+Bttxt$+"]"+Chr$(0)
    Lpoke Addrin,Varptr(Alstring$)
    Dpoke Gintin,Button%
    Gemsys 52
    *Done%=Dpeek(Gintout)
  Return
```

In diesem Zusammenhang ist ein kleiner Trick von Interesse. Nämlich, wie man die recht eintönigen Alert-Symbole nach eigenen Wünschen verändern kann. Durch viele verschiedenartige Symbole läßt sich der Zweck der jeweiligen Alert-Box natürlich wesentlich besser zum Ausdruck bringen als durch !, ? und STOP.

Das GEM legt bei Systemstart Kopien bestimmter ROM-Bereiche im RAM ab, da ja viele GEM-Daten veränderbar sein müssen, was im ROM nicht möglich ist. Zu diesen Daten gehören auch die ALERT- und Desktop-Symbole.

Die folgende Prozedur Atari hat nun die Aufgabe, einen bestimmten Zentralpunkt in diesem GEM-Bereich zu finden und fünf Adressen zurückzugeben. Als Zentralpunkt fungiert das Atari-Symbol, das Sie bewundern können, wenn Sie auf dem Desktop unter dem Menü DESK INFO den Menüpunkt "Desktop Info.." anklicken.

Es ist nötig, diesen Punkt zu suchen, da die Lage des GEM im RAM von verschiedenen Umständen abhängig ist. Dies sind vor allem die verschiedenen TOS-Versionen sowie Anzahl und Größe von Programmen, die vor GEM-Start aus dem Auto-Ordner geladen wurden. Accessories können dagegen im allgemeinen keine Veränderung bewirken, da diese ja erst ausgeführt werden, wenn das GEM bereits installiert ist.

Die Suche geht im Normalfall relativ schnell vonstatten, kann jedoch bei vielen vorgeladenen Auto-Ordner-Programmen etwa eine ganze (!?) Sekunde in Anspruch nehmen. Da die Routine jedoch nur einmal am Programmanfang auszuführen ist, nimmt man diese Wartezeit in Hinblick auf den positiven Effekt gern in Kauf.

Die fünf gelieferten Adressen sind folgende:

1. **Adresse des Atari-Symbols. Die Prozedur ist darauf angewiesen, daß das Bit-Image des Atari-Schriftzuges innerhalb des Symbols in Lage und Form unverändert bleibt, da das Schriftzug-Image als Suchkriterium verwendet wird. Außerdem darf dasselbe Image (wohlgemerkt: ich rede nur von dem Schriftzug innerhalb des Atari-Images) an keiner anderen Speicherstelle auftauchen, es sei denn, es liegt definitiv hinter dem Original.**

Das Symbol hat insgesamt ein Bit-Raster von 32 mal 32 Punkten. Ab der Adresse belegt es also mit seinen 32 Rasterzeilen die folgenden 32 Longwords.

2. Startadresse der Desktop-Symbole. Ab dieser Adresse liegen nacheinander fünf Desktop-Icons aus ebenfalls 32 mal 32 Punkten. Hier ist jedoch zu beachten, daß zu einem Symbol immer zwei Images gehören. Da die Icons auf weißem bzw. grau gerastertem Untergrund frei verschiebbar sind, erhalten sie eine Hintergrund-Maske, die die Aufgabe hat, den Hintergrund für das eigentliche Bit-Muster "freizumachen".

   Die Desk-Symbole sind folgendermaßen organisiert:

Adresse	+0	Maske des Disk-Symbols. Dies ist das Symbol auf dem Desktop, das Sie anklicken, wenn Sie z.B. eine Station (Fenster) öffnen möchten.
"	+128	Bit-Image des Disk-Symbols
Adresse	+256	Maske des Ordner-Symbols im Fenster
"	+384	Bit-Image des Ordner-Symbols
Adresse	+512	Maske des Müll-Symbols
"	+640	Bit-Image des Müll-Symbols
Adresse	+768	Maske des PRG-Symbols im Fenster (sog. "Waschmaschine")
"	+896	Bit-Image des PRG-Symbols
Adresse	+1024	Maske des Datei-Symbols im Fenster
"	+1152	Bit-Image des Datei-Symbols

3. Startadresse der Alert-Icons. Dies sind wiederum Icons mit einem 32x32er Bit-Raster. Hier gibt es allerdings keine Hintergrund-Masken, da diese Symbole generell innerhalb einer ALERT-Box, also auf "weißem" Untergrund verwendet werden.

Adresse	+0	Bit-Image des Ausrufungszeichens
Adresse	+128	Bit-Image des STOP-Schilds
Adresse	+256	Bit-Image des Fragezeichens

4. Startadresse der Maus-Def-Strings. Hier finden Sie acht aufeinanderfolgende Maus-Definitions-Strings, die durch DEFMOUSE 0 bis 7 aufgerufen werden.

   Die Definitions-Strings sind im Aufbau mit den unter DEFMOUSE beschriebenen DEFMOUSE-Strings identisch. Ab dieser GEM-Adresse liegen also 8 Mausdatenblöcke (5 Header-Words, 16 Masken-Words und 16 Image-Words) im 74-Byte-Intervall. Die durch DEFMOUSE 2 aufrufbare "Biene" liegt also z.B. ab Mausstartadresse+2*74.

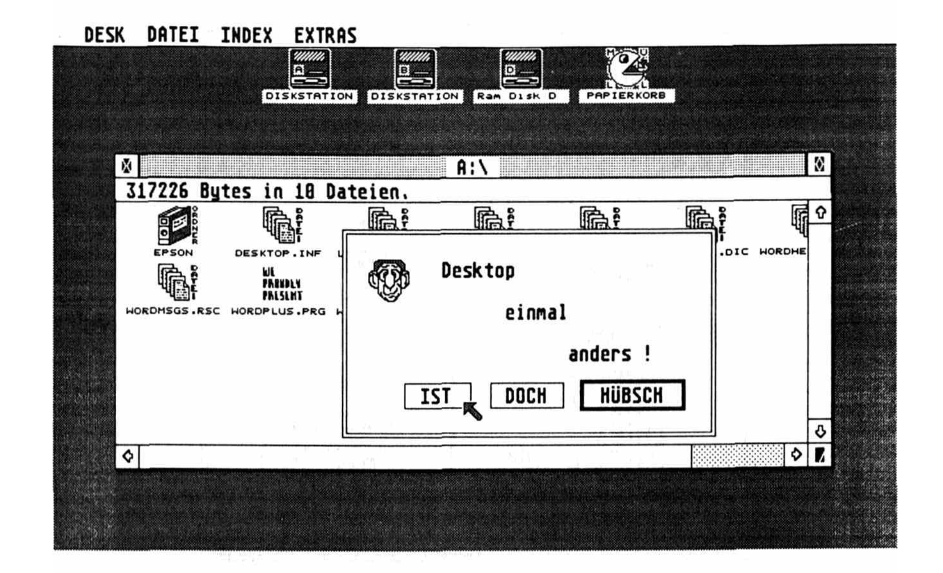
Adresse	+0	Def-String für DEFMOUSE 0
Adresse	+74	Def-String für DEFMOUSE 1
Adresse	+148	Def-String für DEFMOUSE 2
Adresse	+222	Def-String für DEFMOUSE 3
Adresse	+296	Def-String für DEFMOUSE 4
Adresse	+370	Def-String für DEFMOUSE 5
Adresse	+444	Def-String für DEFMOUSE 6
Adresse	+518	Def-String für DEFMOUSE 7

5. Startadresse eines 14-Zeichen-Strings. Der String wird in einer Fileselect-Box über die Index-Zeile mit dem Suchpfad geschrieben. Üblicherweise ist dies die Zeile OBJEKT AUSWAHL. Indem man diese Zeile ersetzt hat man eine Möglichkeit, den Zweck der FILESELECT-Box zu kommentieren. Dabei ist jedoch streng darauf zu achten, daß der an diese Adresse geschriebene String nicht mehr als 14 Zeichen lang ist.

```
Atari(*Atari%,*Dsymbols%,*Asymbols%,*Msymbols%,*Fcomm%)
 For I%=0 To 31                    ! Je 32 Bit-Zeilen
   Lpoke Xbios(2)+I%*80,Lpeek(Atari%+I%*4)
   ' Atari-Symbol auf den Bildschirm schreiben
```

```
    Lpoke Xbios(2)+2560+I%*80,Lpeek(Dsymbols%+128+I%*4)
    ' Disketten-Symbol auf den Bildschirm schreiben
    Lpoke Xbios(2)+5120+I%*80,Lpeek(Asymbols%+I%*4)
    ' Alert-Ausrufungszeichen auf den Bildschirm schreiben
    Dpoke Xbios(2)+7920+I%*80,Dpeek(Msymbols%+10+I%*2)
    ' Mauspfeil+Maske auf den Bildschirm schreiben
  Next I%
Print At(1,7);
For I%=0 To 13                      ! 14 Zeichen FILESELECT-String
  Print Chr$(Peek(Fcomm%+I%));      ! Lesen und schreiben
Next I%
'
C$=Space$(74)                       ! Maus-Image-Puffer
Bmove Msymbols%,Varptr(C$),74       ! Alte Maus retten
A$=Mki$(1)+Mki$(1)+Mkl$(1)+Mki$(0)  ! Maus-Header bilden und..
Bmove Varptr(A$),Msymbols%,10       ! ..Original-Header überschreiben
For I%=0 To 31                      ! 16 Masken- und 16 Image-Zeilen
  Read A%                           ! aus DATAs lesen
  Dpoke Msymbols%+10+I%*2,A%        ! und den Standard-Mauspfeil..
Next I%                             !                ..überschreiben
Data 65520,65520,65520,65504,65472,65504,65520,65528
Data 65532,65534,63487,58367,511,255,127,62
Data 0,32736,27328,21888,27392,21888,27328,23904
Data 30384,25432,16812,214,106,54,28,0
'
For I%=0 To 31                      ! 32-Bit-Zeilen
  Read A%                           ! aus DATAs lesen
  Lpoke Asymbols%+I%*4,A%           ! und das Alert-Ausrufungs-
Next I%                             ! zeichen überschreiben
Data 16777152,64312672,113245872,91226968,111393192,98438872
Data 109052008,94322264,523314046,828848611,1305162991
Data 2077105401,1774980301,1221332165,1220807109,1219758917
Data 1218186821,1820592205,646187851,814619078,478884044
Data 126476920,8929856,8930880,11013184,11303232,6783360
Data 3160832,1369600,919040,461824,129024
Alert 1,"Neues ALERT-Symbol|u. neuer Zeiger !",1,"OKAY",B%
'
A$=" * KOMMENTAR *"                 ! 14 Zeichen FILESELECT-
'                                   ! Kommentar in den
Bmove Varptr(A$),Fcomm%,14          ! GEM-String-Puffer schreiben
Fileselect "\*.*","",B$             ! FILESELECT-Box
'
For I%=0 To 63                      ! 64-Bit-Zeilen (Maske u. Image)
  Read A%                           ! aus DATAs lesen und das
  Lpoke Dsymbols%+I%*4,A%           ! Disk-Symbol überschreiben
Next I%
Data 2147483646,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
Data -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,2147483646
Data 0,1073741820,1572264822,2032439902,1380525206,1955744062
Data 1495568982,1917396126,1418873142,2109135742,1610612726
Data 1789569710,1431655766,1611311790,1343575382,1611311790
Data 1343575382,1611399166,1343619126,1611661294,1431655766
Data 2147483646,1073741830,1074265862,1341654006,1341129718
Data 1075837958,2147483646,1073741838,1073741822,536870908,0  '
Bmove Dsymbols%+128,Asymbols%+128,128 ! 128 Bytes aus dem
'                                   ! Disk-Image-Puffer in den
'                                   ! Alert-Image-Puffer schreiben
A$="Wenn Sie zum Desktop|zurückkehren, wird das|"
A$=A$+"Disketten-Symbol verändert|sein!"
```

```
Alert 2,A$,1,"OKAY",B%            ! ALERT-Box aufrufen
'
Bmove Varptr(C$),Msymbols%,74     ! Alte Maus restaurieren
'
Procedure Atari(Aptr1%,Aptr2%,Aptr3%,Aptr4%,Aptr5%)
  ' Aptr1% = Pointer auf 4-Byte-Integer-Rückgabevariable, die
  '          nach Abschluß die Atari-Symbol-Adresse enthält.
  ' Aptr2% = Pointer auf 4-Byte-Integer-Rückgabevariable, die
  '          nach Abschluß die Startadresse der Desk-Symbole
  '          enthält.
  ' Aptr3% = Pointer auf 4-Byte-Integer-Rückgabevariable, die
  '          nach Abschluß die Startadresse der Alert-Symbole
  '          enthält.
  ' Aptr4% = Pointer auf 4-Byte-Integer-Rückgabevariable, die
  '          nach Abschluß die Startadresse der Maus-Images
  '          enthält.
  ' Aptr5% = Pointer auf 4-Byte-Integer-Rückgabevariable, die
  '          nach Abschluß die Adresse des FILESELECT-Kommentars
  '          enthält.
  Local Atari$,Blen%,Ipos%,Ipos2%,Buff$,Acnt%
  Atari$=Mkl$(&H9F90F8C)          !---------------.
  Atari$=Atari$+Mkl$(&H1DFB8FCC)+Mkl$(&H1C638CEC) !  Bit-Image
  Atari$=Atari$+Mkl$(&H3666CCEC)+Mkl$(&H3666CDCC) !- des Atari-
  Atari$=Atari$+Mkl$(&H7F6FED8C)+Mkl$(&H7F6FEDCC) !  Schriftzugs
  Atari$=Atari$+Mkl$(&H636C6CEC)+Mkl$(&H636C6C6C) !
  '                               !---------------'
  Blen%=Min(Fre(0)-1000,32000)     ! max. Suchblocklänge
  Buff$=Space$(Blen%)              ! Suchblockpuffer vorbereiten
  For Acnt%=0 To Int(200000/Blen%)+1 ! Suchblockanzahl
    Bmove 10000+Acnt%*(Blen%-40),Varptr(Buff$),Blen% ! Suchblock
    '                              ! in Puffer übertragen
    Ipos%=Instr(Buff$,Atari$)      ! Atari-Schrift-Image suchen
    If Ipos%                       ! Gefunden?
      *Aptr1%=10000+Acnt%*(Blen%-40)+Ipos%-1-88   ! Offset 1
      *Aptr2%=10000+Acnt%*(Blen%-40)+Ipos%-1+3486 ! Offset 2
      *Aptr3%=10000+Acnt%*(Blen%-40)+Ipos%-1-7086 ! Offset 3
      *Aptr4%=10000+Acnt%*(Blen%-40)+Ipos%-1-6702 ! Offset 4
      *Aptr5%=10000+Acnt%*(Blen%-40)+Ipos%-1-9304 ! Offset 5
    Endif
    Exit if Ipos%
  Next Acnt%
Return
```



FILESELECT { FILESE } — Datei auswählen

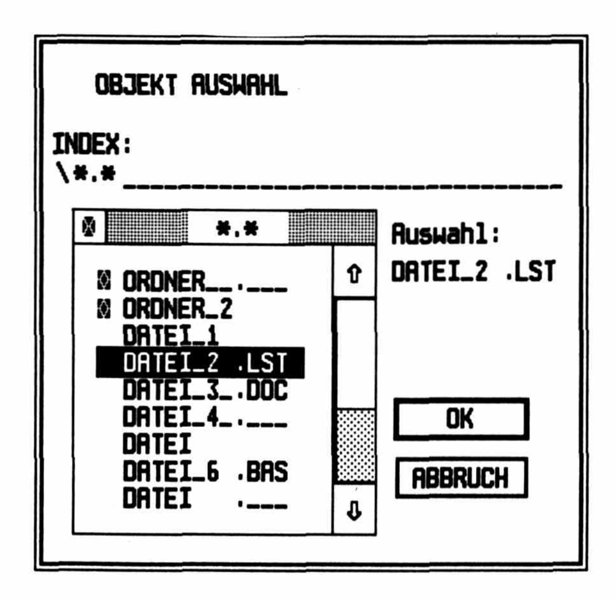
In V3.0: { FILE }

FILESELECT "Pfad","Auswahl",Backvar$

Erstellt ein Dialog-Formular zur Dateiauswahl und liefert den gewählten Dateinamen (ggfs. inkl. Pfad).

Bei Rückkehr zum Programm sind vier verschiedene Eintragsmöglichkeiten in Backvar$ möglich:

1.	Wurde vom Anwender eine Datei gewählt, steht ihr Name auch anschließend in Backvar$.
2.	Wurde die OK-Box ohne Auswahl bedient, gibt es zwei Varianten:
2a.	Es wurde durch Auswahl ein Dateiname übergeben, der noch in der Eintragszeile steht und die Ok-Box bedient, dann steht dieser Name auch in Backvar$.
2b.	Es wurde keine Auswahl getroffen und auch kein Name übergeben, bzw. die Eintragszeile wurde vom Anwender gelöscht und die OK-Box wurde bedient, dann wird in Backvar$ ein Backslash (\) geliefert.
3.	Wird Abbruch angeklickt, ist Backvar$ absolut leer.



Dies ist einer der wichtigsten Befehle zur Bewältigung von Diskettenoperationen. Die Menge an Funktionen, die dieser so einfach scheinende Befehl in sich vereinigt, kann man nur ermessen, wenn man schon einmal versucht hat, auf die herkömmliche Weise Dateien zu speichern, zu laden, zu löschen oder zu verändern.

Diese Box listet in einem Fenster die unter dem angegebenen Pfadnamen (Zeile über dem Fenster) verfügbaren Dateien auf. Die links neben dem Namen gekennzeichneten Dateien stellen sogenannte Subdirectories (engl.: Unter-Inhaltsverzeichnis) dar, in denen weitere Dateien unter dem für dieses Subdirectory typischen Pfadnamen abgespeichert sein können. Durch Anklicken des Sub-Dir-Namens wird es geöffnet und die darin befindlichen Dateien können angewählt werden. Verlassen wird ein Unter-Verzeichnis, indem man das kleine Feld links neben dem Kopfbalken anklickt.

Da GFA-BASIC immer die Disk-Station zur aktuellen erklärt, von welcher das Programm geladen wurde, kann es notwendig werden, im Pfadnamen die Stationsangabe zu ändern. Sie können dazu mit dem Mauszeiger in die Zeile unter INDEX klicken. Der Strich-Cursor begibt sich dann in diese Zeile und Sie können nun durch die Tasten <Esc>, <Backspace>, <Delete> oder die <Cursor>-Tasten die Zeile oder einzelne Zeichen löschen oder den Cursor an eine bestimmte Stelle bewegen. Haben Sie das getan, können Sie nun die Pfadbezeichnung ändern und sich so z.B. den Inhalt einer anderen Diskette anzeigen

lassen. Nach der Änderung brauchen Sie jetzt nur noch den grauen Balken über dem Directory-Fenster anzuklicken und das neue Directory wird angezeigt.

Es folgt eine fast unscheinbare Routine, die es aber in sich hat. Wie oft stellt sich die Aufgabe, die Richtigkeit einer Dateinamen-Eingabe zu überprüfen oder ein Backup-File anzulegen. Während der eigentliche Dateiname in den allermeisten Fällen für den Anwender frei bestimmbar ist, hat man als Programmierer doch oft ein Interesse daran, daß die Extension richtig gesetzt ist. Es nützt meist wenig, die Extension mit dem zweiten Parameter-String der FILESELECT-Box vorzugeben, wenn anschließend bei der Eingabe die Extension gelöscht oder verändert wird. Um sicherzustellen, daß garantiert die richtige Extension verwendet wird, kann man nun diese Prozedur folgendermaßen aufrufen:

```
Fileselect "\*.ABC",".ABC",A$  ! Eingabe des Dateinamens
@Extend(A$,"ABC",*N_file$)     ! Extension überprüfen
Print N_file$
'
Procedure Extend(Pr$,Ex$,Ps%)
  ' Pr$ = Dateipfad u. -name
  ' Ex$ = gewünschte Extension
  ' Ps% = String-Rückgabe
  '
  Local Nl%,Dn$,I%
  If Right$(Pr$)<>"\" And Right$(Pr$,5)<>"\."+Ex$ And Pr$>""
    '                          ! Gültiger Dateiname?
    For I%=Len(Pr$) Downto 1   ! Namen von hinten aus nach dem
      '                        ! erstem Backslash durchsuchen
      Inc Nl%                  ! Zeichen mitzählen
      Exit if Mid$(Pr$,I%,1)="\" ! Exit, wenn Backslash erreicht
    Next I%
    Dn$=Right$(Pr$,Nl%)        ! Reinen Dateinamen bilden
    If Instr(Dn$,".")=0        ! Keine Extension enthalten?
      *Ps%=Pr$+"."+Ex$         ! Pfad und Extension zurückgeben
      Dn$=Dn$+"."+Ex$          ! Namen komplettieren
    Endif
    If Right$(Dn$,4)<>"."+Ex$  ! Falsche Extension?
      If Left$(Dn$,2)<>"\."    ! Name größer als nur Extension?
        *Ps%=Left$(Pr$,Len(Pr$)-Nl%)+Left$(Dn$,Instr(Dn$,"."))+Ex$
          '                      ! Pfad + Name + Extension zurück
      Else                     ! Name besteht nur aus Extension!
        *Ps%="000"             ! Unbrauchbaren Namen zurückgeben
      Endif
    Else                       ! Richtige Extension!
      *Ps%=Left$(Pr$,Len(Pr$)-Nl%)+Dn$ ! Pfad + Name zurück
    Endif
  Else                         ! Ungültiger Dateiname!
*Ps%="000"                 ! Unbrauchbaren Namen zurückgeben
  Endif
Return
```

Wurde in diesem Beispiel in der FILESELECT-Box ohne Änderung einfach die OK- oder ABBRUCH-Box angeklickt, erhält man in N_file$ nach Extend-Aufruf den Ausdruck 000. Das gleiche geschieht, wenn vom Anwender entweder die Auswahlzeile ersatzlos gelöscht wurde (<Esc>) oder nur die Extension geändert, aber kein Name dazu eingegeben wurde. In allen anderen Fällen wird die Extension des eingegebenen Namens mit der Vorgabe im zweiten Parameter-String des Extend-Aufrufs verglichen und bei Nicht-Übereinstimmung durch die Vorgabe ersetzt. Anschließend erhalten Sie dann den gesamten Dateinamen mit evtl. geänderter Extension in N_file$ zurück.

Bei Backup-Files kommt noch eine zweite Mini-Routine zum Einsatz. Die Auswirkung einer Backup-Routine begegnet Ihnen immer dann, wenn Sie ein .GFA- oder .LST-File abspeichern, dessen Name bereits auf der Diskette exisitiert. GFA-BASIC ändert dann automatisch die Extension der bereits bestehenden Datei in .BAK um.

Diese Routine Backup erwartet zwei Parameter-Strings. Der erste gibt den Namen an, unter welchem die Datei abgelegt werden soll. Der zweite bestimmt die Extension, die Sie jener Datei geben wollen, die evtl. unter demselben Namen wie die neue Datei schon auf Diskette existiert und nun als Backup gesichert werden soll.

Die Backup-Routine erledigt nun das Finden und Umbenennen mit Hilfe der Extend-Routine, so daß Sie anschließend Ihre neue Datei anlegen können. Als Beispiel lege ich hier das obere Viertel des Bildschirms auf Diskette ab. Der Backup-Effekt wird erst deutlich, wenn Sie das Programm zweimal starten und sich die Dateien mit FILES im Direktmodus anschauen. Sie haben nun zwei Dateien mit gleichem Namen, aber unterschiedlicher Extension.

```
Fileselect "\*.SCR",".SCR",A$ ! Eingabe des Dateinamens
@Backup(A$,"BAK")             ! Evtl. Backup anlegen
@Extend(A$,"SCR",*N$)         ! Extension überprüfen
If N$<>"000"                  ! Brauchbarer Pfad + Name?
  Bsave N$,Xbios(2),8000      ! Datei anlegen
Endif
'
Procedure Backup(Pr$,Ex$)
  ' Pr$ = Dateipfad u. -name
  ' Ex$ = Backup-Datei-Extension
  Local Xn$
    If Exist(Pr$)             ! Datei auf Diskette vorhanden?
      @Extend(Pr$,Ex$,*Xn$)   ! Backup-Extension einbauen
      If Xn$<>"000"           ! Brauchbarer Dateiname?
        If Exist(Xn$)         ! Schon Backup-File vorhanden?
          Kill Xn$            ! Dann löschen
        Endif
        Name Pr$ As Xn$   ! Alte Datei auf Backup umbenennen
```

```
    Endif
  Endif
Return
```

Ein weiteres - schon bei ALERT angedeutetes - Problem im Umgang mit der FILESELECT-Box ist, die Box bzw. den Zweck Ihres Aufrufs zu kommentieren. Aus vielen kommerziellen Programmen kennt man diesen grauen Textbalken, der oberhalb der Box angezeigt wird, um dem Benutzer Informationen darüber zu bieten, was er denn nun mit der FILESELECT-Box auswählen soll. Dieser Balken sieht recht zweckmäßig aus und ist es auch. Nur ist es nicht immer einfach, den gewünschten Kommentartext innerhalb der Box bzw. die Textbox oberhalb der FILESELECT-Box zu zentrieren, da sich durch die verschiedenen Bildschirm-Auflösungen jeweils andere Offsets ergeben.

Die folgende Prozedur erledigt diese Aufgaben in allen drei Auflösungen. Innerhalb der Routine wird GRAPHMODE 1, DEFTEXT 1,0,,13 und DEFFILL 1,0,0 eingestellt. Vergessen Sie also nicht, ggfs. hinterher die vor Aufruf gültigen Einstellungen zu restaurieren.

```
Xrs%=2-Sgn(Xbios(4))            ! X-Auflösungsteiler
Yrs%=Min(2,3-Xbios(4))          ! Y-Auflösungsteiler
Deffill ,2,4                    ! DEFFILL grau
Pbox 10/Xrs%,10/Yrs%,630/Xrs%,390/Yrs% ! Hintergrund zeichnen
@Head(1,"KOMMENTAR ZUR FILE-AUSWAHL")  ! Head aufrufen
Fileselect "\*.*","",A$                ! Datei-Auswahl
@Head(0,"")                            ! Head wieder löschen
Procedure Head(H.flg%,H.txt$)
  ' Zeichnet eine graue Box mit vorgegebenen Text über
  ' eine FILESELECT-Box (in Hires/Midres/Lowres).
  ' H.flg% = Flag, gibt an, ob die Box gezeichnet (1)
  '          oder gelöscht (0) werden soll. Wurde Head mit
  '          H.flg% = 1 zwei- oder mehrmals hintereinander
  '          aufgerufen, ohne jedesmal hinterher die Box mit
  '          H.flg% = 0 wieder zu löschen, kann der Hintergrund
  '          nicht mehr retauriert werden.
  ' H.txt$ = Kommentartext zur Fileselect-Box (max. 38 Zeichen)
  Local H.res%,H.hlf%,H.dis%,H.re%,H.xr%,H.yr%,H.bxr%
  H.txt$=Left$(H.txt$,Min(38,Len(H.txt$)))  Textlänge trimmen
  H.xr%=2-Sgn(Xbios(4))   ! Lokaler X-Auflösungsteiler
  H.yr%=Min(2,3-Xbios(4)) ! Lokaler Y-Auflösungsteiler
  If H.xr%<>2             ! Kein Lowres?
    H.re%=1               ! Box-X-Offset-Multiplikator
  Endif
  If H.flg%=1             ! Text-Box darstellen?
    H.hlf%=Len(H.txt$)*4  ! Halbe Textlänge in Pixel
    H.dis%=2-H.yr%        ! Box-Y-Offset-Multiplikator für Hires
    If H.xr%<>2           ! In Hires/Midres?
      H.bxr%=481          ! Rechte Box-Koordinate setzen
    Else                  ! In Lowres
      H.bxr%=319          ! Rechte Box-Koordinate setzen
    Endif
    Graphmode 1           ! GRAPHMODE 1
```

```
      Deftext 1,0,,13        ! Standard-Text einschalten
      Get 158*H.re%,0,H.bxr%,399/H.yr%/6,H.bg$ ! Hintergrund sichern
      Deffill 1,2,2          ! DEFFILL hellgrau
      Pbox 158*H.re%,16/H.yr%+H.dis%*16,H.bxr%,...        ! Box
              ...(50/H.yr%-H.dis%*12)+H.dis%*16        ! zeichnen
      Deffill 1,0,0          ! DEFFILL weiß
      Pbox 320/H.xr%-H.hlf%-2,16/H.yr%+H.dis%*16,320/H.xr%...
              ...+H.hlf%+2,(50/H.yr%-H.dis%*12)+H.dis%*16
      '                      ! Textbalken zeichnen
      Text 320/H.xr%-H.hlf%,(44/H.yr%-H.dis%*10)+H.dis%*16,H.txt$
      '                      ! Text zentriert in Box einsetzen
    Else                     ! Text-Box löschen!
      Put 158*H.re%,0,H.bg$ ! Hintergrund restaurieren
      Clr H.bg$              ! Hintergrund-Puffer löschen
    Endif
  Return
```

MOUSE { M } Maus-Status ermitteln (gesamt)

V3.0: { MOU }

MOUSE Xpos,Ypos,Tasten

In Xpos und Ypos wird die aktuelle X-, bzw. Y-Koordinate des Mauszeigers, sowie in Tasten der Status der Maustasten übergeben.

```
0 = Keine Taste gedrückt
1 = Linke Taste gedrückt    (Bit 0   -> 2^0     = 1)
2 = Rechte Taste gedrückt   (Bit 1   -> 2^1     = 2)
3 = Beide Tasten gedrückt   (Bit 0+1 -> 2^0+2^1 = 3)
```

Als Bit-Vektor betrachtet, sind von Tasten nur die untersten 2 Bits interessant. Dabei steht Bit 0 für die linke Maustaste und Bit 1 für die rechte.

Anmerkung: In Version V3.0 werden bei geöffneten GEM-Windows (bzw. bei CLIP OFFSET) in Xpos und/oder Ypos auch negative Werte geliefert, wenn sich der Mauszeiger links und/oder oberhalb des Windows (bzw. des CLIP-Nullpunkts) befindet.

Beispiel:

Im Zusammenhang mit der Maus stellt sich immer wieder das Problem, einen Bildschirmbereich zu umrahmen, um ihn für spätere Zwecke zu definieren oder zu kennzeichnen. In V3.0 kann dazu leicht die AES-Funktion GRAF_RUBBERBOX() verwendet werden. Diese hat allerdings drei Nachteile. Der erste ist, daß die Rahmenbox in einem Punkt-Linienmuster gezeichnet wird und so nicht auf jedem Untergrund leicht zu erkennen ist. Der zweite besteht darin, daß bei

der Box-Definition nur die linke Maustaste zugelassen ist und als dritten Nachteil empfinde ich, daß aufgrund bestimmter GEM-interner Gründe der Mausklick bei Aufruf nicht immer registriert wird, so daß die Maustaste schon als losgelassen betrachtet wird, bevor die Box überhaupt definiert werden konnte.

Aus diesem Grund habe ich eine Prozedur entworfen, die Ihnen weitgehend freie Hand läßt und die nach eigenem Bedarf verändert oder erweitert werden kann. Diese Routine stellt allerdings DEFLINE 0,0,0,0, COLOR 1 und GRAPHMODE 1 ein, so daß ggfs. nach Aufruf die vorher gültigen Einstellungen restauriert werden müssen. Sie können diese Einstellungen natürlich auch in der Routine verändern, um andere Effekte zu erzielen (z.B. in GRAPHMODE 3 oder COLOR 0).

Grundsätzlich funktioniert die Prozedur exakt genauso wie die AES-Funktion. Hier ist es allerdings gleichgültig, ob die linke, die rechte oder beide Maustasten gedrückt wurden. Ebenso wie die AES-Funktion **muß** die Prozedur mit gedrückter Maustaste aufgerufen werden, da Sie sonst sofort wieder beendet wird. Ein wesentlicher Unterschied ist, daß diese Prozedur auch negative Minimalwerte verarbeitet, daß heißt, die Koordinaten der Endposition des Mauspfeils können kleiner sein als die Koordinaten der Startposition. In diesem Fall erhält man in den beiden Rückgabevariablen auch negative Box-Ausmaße.

```
Deffill ,2,4                 ! DEFFILL grau
Pbox 10,10,200,200           ! Hintergrund zeichnen
Do                           ! Endlos-Schleife
  If Mousek                  ! Maustaste gedrückt?
    Mouse X%,Y%,K%           ! Startkoordinaten holen
    Rubberbox(X%,Y%,-30,-30,*Xx%,*Yy%) ! Aufruf
    Box X%,Y%,X%+Xx%,Y%+Yy%! Box zeichnen
  Endif
Loop
'
Procedure Rubberbox(Xp%,Yp%,Xmin%,Ymin%,Xret%,Yret%)
  ' Xp%   = X-Startkoordinate
  ' Yp%   = Y-Startkoordinate
  ' Xmin% = Minimale Breite der Gummi-Box (auch neg.)
  ' Ymin% = Minimale Höhe der Gummi-Box (auch neg.)
  ' Xret% = Pointer auf 4-Byte-Integer, die nach Abschluß
  '         die letzte Breite der Box enthält (auch neg.)
  ' Yret% = Pointer auf 4-Byte-Integer, die nach Abschluß
  '         die letzte Höhe der Box enthält (auch neg.)
  Local Sc1$,Sc2$,Sc3$,Sc4$,Mx1%,Mx2%,Mx2%,My2%,Mk%
  Defline 0,0,0,0            ! DEFLINE voll/dünn
  Color 1                    ! COLOR schwarz
  Graphmode 1                ! Replace-Modus
  Repeat
    Mouse Mx1%,My1%,Mk%      ! Neue Koordinaten holen
    Mx2%=Max(Xp%+Xmin%,Mx1%) ! X-Koordinate auf Minimum trimmen
```

```
        My2%=Max(Yp%+Ymin%,My1%) ! Y-Koordinate auf Minimum trimmen
        Get Min(xp%,Mx2%),Min(yp%,My2%),...    !--. Es wird der
        ...Max(Mx2%,xp%),Min(yp%,My2%),Sc1$      ! Hintergrund
        Get Max(xp%,Mx2%),Min(yp%,My2%),...      ! jeder Box-Linie
        ...Max(Mx2%,xp%),Max(yp%,My2%),Sc2$      ! einzeln
        Get Min(xp%,Mx2%),Max(yp%,My2%),...      ! gesichert
        ...Max(Mx2%,xp%),Max(yp%,My2%),Sc3$      ! (Speicherplatz-
        Get Min(xp%,Mx2%),Min(yp%,My2%),...      ! Ersparnis!)
        ...Min(Mx2%,xp%),Max(yp%,My2%),Sc4$      !
        Box Xp%,Yp%,Mx2%,My2%                  !--'
        Repeat                                 ! Warte...
          On menu  ! Ereignis-Überwachung !
        Until Mousex<>Mx1% Or Mousey<>My1% Or Mousek=0
        ' ...bis die Maus bewegt oder ein Mausknopf gedrückt wird
        Put Min(Xp%,Mx2%),Min(Yp%,My2%),Sc4$  !--.
        Put Min(Xp%,Mx2%),Max(Yp%,My2%),Sc3$    ! Box-Hintergrund
        Put Max(Xp%,Mx2%),Min(Yp%,My2%),Sc2$    ! restaurieren
        Put Min(Xp%,Mx2%),Min(Yp%,My2%),Sc1$  !--'
      Until Mk%=0                  ! Exit, wenn Maustaste = 0
      *Xret%=Mx2%-Xp%              ! Letzte Breite der Box zurück
      *Yret%=My2%-Yp%              ! Letzte Höhe der Box zurück
    Return
```

Weitere Beispiele zur Maus-Abfrage finden Sie in Hülle und Fülle im Buch verteilt.

MOUSEX, MOUSEY, MOUSEK

Maus-Status ermitteln (einzeln)

Var=MOUSEX **=> X-Position**
Var=MOUSEY **=> Y-Position**
Var=MOUSEK **=> Maustastenstatus (siehe MOUSE)**

Einzelabfrage-Funktionen für den jeweils gewünschten Maus-Status. In Version V3.0 siehe Anmerkung zu MOUSE.

Version 3.0

SETMOUSE { SETM }

Maus simulieren

SETMOUSE Xpos,Ypos [,Button]

Setzt den Mauszeiger auf die Position mit den Koordinaten Xpos/Ypos und schaltet ihn ein. Wird optional der Parameter Button verwendet, wird dadurch das Drücken der darin angegeben Maustaste (0 - 3) simuliert (funktioniert in den AES nur eingeschränkt).

Als V2.xx-Anwender beachten Sie bitte die Setmouse-Prozedur unter PSET. Das dortige Beispiel kann auch für den SETMOUSE-Befehl dienen.

Wer es möchte oder für notwendig hält, kann mit diesem Befehl den Mauszeiger auch zu einem zusätzlichen Sprite umfunktionieren:

```
A$=Mkl$(0)+Mkl$(1)+Mki$(0)+String$(16,Mkl$(-34953))
Do
  For I%=0 To 319
    Vsync
    Sprite A$,I%,100
    Showm
    Setmouse I%,120
  Next I%
Loop
```

Version 3.0

STICK { STI } Maus-Port-Abfragemodus bestimmen

STICK(Modus) **- Befehl -**

Bestimmt den Abfragemodus der Maus-/Joystick-Ports.

Modus:

0	Mausmodus (für V2.xx: entspricht OUT 4,8).
1	Joystick-Modus (für V2.xx: entspricht OUT 4,20).

MOUSE-Befehle (in V2.xx und V3.0) und STICK() schalten automatisch den entsprechenden Modus ein, wodurch der Befehl STICK in den meisten Fällen überflüssig ist. Ist bei Programmende oder vor AES-Aufrufen (FILESELECT, ALERT etc.) STICK 1 aktiv, sollte STICK 0 (oder OUT 4,8) verfügt werden, da sonst die Mausmeldung ausbleibt.

Bei ALERT-Boxen, die keinen Default-Button vorsehen, kann das dann fatale Folgen haben, wenn aus irgendwelchen Gründen (normalerweise unmöglich) die Maus-Steuerung über die <Alternate>- in Verbindung mit den Cursor-Block-<Tasten> gestört sein sollte. Im Normalfall kann jedoch die Maus auch bei aktivem STICK 1 über die <Alternate>-Taste gesteuert werden:

<Alternate><Pfeil-links>	Maus schnell nach links.
<Alternate><Shift><Pfeil-links>	Maus langsam nach links.
<Alternate><Pfeil-rechts>	Maus schnell nach rechts.
<Alternate><Shift><Pfeil-rechts>	Maus langsam nach rechts.
<Alternate><Pfeil-hoch>	Maus schnell nach oben.
<Alternate><Shift><Pfeil-hoch>	Maus langsam nach oben.
<Alternate><Pfeil-runter>	Maus schnell nach unten.
<Alternate><Shift><Pfeil-runter>	Maus langsam nach unten.
<Alternate><Insert>	Entspr. linkem Mausknopf.
<Alternate><ClrHome>	Entspr. rechtem Mausknopf.

Ein Beispiel finden Sie unten unter STICK().

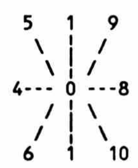
Version 3.0

STICK() **Maus-Port im Joystick-Modus abfragen**

Var=STICK(Maus-Port) **- Funktion -**

Ermittelt den aktuellen Status des angegebenen Maus-Ports (0 oder 1). STICK(1) kann auch abgefragt werden, wenn durch den STICK-Befehl **nicht** der Joystick-Modus eingeschaltet ist. STICK(0) liefert dagegen **nur** im Joystick-Modus brauchbare Werte.

Es werden folgende Werte geliefert:



Als Bit-Vektor betrachtet, sind von STICK() nur die untersten 4 Bits interessant:

Bit 0	Joystick hoch.
Bit 1	Joystick runter.
Bit 2	Joystick links.
Bit 3	Joystick rechts.

Für V2.xx:

```
Var=@Stick(Maus-Port)
DEFFN Stick(Mp%)=Peek(3592+Mp%)
' Im Gegensatz zur V3.0-Funktion muß bei
' dieser DEFFN-Funktion für den Port 0
' erst OUT 4,20 eingesetzt werden.
' Weiterhin liefert diese Funktion auch
' gleichzeitig den Status des Fire-Buttons
' (siehe STRIG()). Wurde er gedrückt, erhöht
' sich der oben beschriebene Joystick-Wert
' um den Wert 128. D.h, sobald der Fire-Button
' gedrückt wird, ist das höchste Bit (Bit 7)
' des LO-Bytes von Var gesetzt.
```

Beispiel: Dieses Beispiel erwartet, daß ein Joystick in Port 0 (eigentlich Maus-Port) angeschlossen ist.

```
Circle 7,7,6           ! --.
Circle 7,7,4               !
```

```
Line 1,7,13,7           !- Pseudo-Sprite bauen
Line 7,1,7,13           !
Get 0,0,15,15,A$       !---'
X%=160                 ! X-Startkoordinate
Y%=100                 ! Y-Startkoordinate
On Break Gosub Ende    ! Break-Funktion abfangen
Stick 1                ! Joystick-Modus einschalten
' in V2.xx: OUT 4,20 !    "      "      "
Do
  Print At(1,1);"  ";At(1,1);
  If Stick(0) And 1    ! Joystick nach oben?
    ' in V2.xx: If Peek(3592) And 1
    Y%=Max(0,Y%-1)     ! Y-Koordinate vermindern
    Out 5,1            ! Hoch-Pfeil zeigen
  Endif
  If Stick(0) And 2    ! Joystick nach unten?
    ' in V2.xx: If Peek(3592) And 2
    Y%=Min(199,Y%+1)   ! Y-Koordinate erhöhen
    Out 5,2            ! Runter-Pfeil zeigen
  Endif
  If Stick(0) And 4    ! Joystick nach links?
    ' in V2.xx: If Peek(3592) And 4
    X%=Max(0,X%-1)     ! X-Koordinate vermindern
    Out 5,4            ! Links-Pfeil zeigen
  Endif
  If Stick(0) And 8    ! Joystick nach rechts?
    ' in V2.xx: If Peek(3592) And 8
    X%=Min(319,X%+1)   ! X-Koordinate erhöhen
    Out 5,3            ! Rechts-Pfeil zeigen
  Endif
  Vsync
  If Strig(0)          ! Fire-Button gedrückt?
  ' in V2.xx: If Peek(3592) And 128
    Print At(1,1);"Feuer "
    For I%=0 To 6      ! -------.
      Circle X%+7,Y%+7,I%     !- Grafischer Effekt
    Next I%            ! -------'
    Print At(1,1);"      "
  Endif
  Put X%,Y%,A$         ! Pseudo-Sprite setzen
Loop
Procedure Ende
  ' Diese Prozedur und das oben eingesetzte ON BREAK GOSUB
  ' kann weggelassen werden, wenn nach Programmende im
  ' Direktmodus STICK 0 (bzw. in V2.xx: OUT 4,8) eingegeben
  ' wird. Geschieht dies nicht, erfolgt keine Mausmeldung.
  Stick 0              ! Mausmodus wieder einschalten
  ' in V2.xx: OUT 4,8 ! "      "      "      "
  Edit
Return
```

Version 3.0

STRIG() — Joystick-Fire-Buttons abfragen

Var=STRIG(Maus-Port)

Ermittelt den aktuellen Status des Fire-Buttons am angegebenen Maus-Port (gedrückt = 1, nicht gedrückt = 0).

Ein Beispiel finden Sie oben unter STICK().

14. Window-Programmierung mit BASIC-Befehlen

CLEARW { CLE W } Fenster-Inhalt löschen

CLEARW Nummer
CLEARW [#]Nummer **(nur V3.0)**

Das Fenster, dessen Nummer (0 - 4) hiermit übergeben wird, wird gelöscht. Der Befehl CLS kann ebenfalls verwendet werden. Allerdings kann dadurch nur das jeweils aktuelle Window gelöscht werden.

Das Löschen des Fensterinhaltes ist immer dann notwendig, wenn das Fenster bewegt oder verkleinert wurde. Sie werden feststellen, daß bei Bewegungen oder Größenänderungen der Fenster die Inhalte der übrigen Fenster dabei überzeichnet werden. Die Verwaltung der jeweiligen Fensterinhalte ist allein Ihre Aufgabe, da niemand außer Ihnen wissen kann, was mit den Inhalten bei bestimmten Zuständen geschehen soll.

PRINT- sowie sonstige Text-Ausgaben können leicht durch eine Schleife restauriert werden, die die Textinhalte des Fensters neu in das Fenster hineinschreibt. Dagegen ist die Restauration von Grafik-Inhalten allerdings eine recht komplizierte Angelegenheit. Beachten Sie dazu WIND_GET().

Ein Beispiel für die Window-Befehle finden Sie am Ende des Kapitels.

CLOSEW { CL W } Fenster schließen

CLOSEW Nummer
CLOSEW [#]Nummer **(nur V3.0)**

Das Fenster mit der angegebenen Window-Nummer wird geschlossen.

Ein Beispiel für die Window-Befehle finden Sie am Ende des Kapitels.

FULLW { FU } Fenster auf maximale Größe bringen

In V3.0: { FUL }

FULLW Nummer
FULLW [#]Nummer **(nur V3.0)**

Nummer bestimmt die Nummer des Windows (1 - 4), welches bis an die Menüzeilen-Aussparung am oberen Bildrand vergrößert werden soll.

Am Ende des Kapitels finden Sie ein Beispiel für alle Window-Befehle.

INFOW { INF } Fenster-Informationszeile bestimmen

INFOW Nummer,"Text"
INFOW [#]Nummer,"Text" **(nur V3.0)**

Ein GEM-Window kann eine Informationszeile erhalten, die sich dann direkt unterhalb der Titelzeile (grauer MOVE-Balken) befindet. Nummer bestimmt die Nummer des anzusprechenden Windows.

Text enthält den gewünschten Text der Info-Zeile und kann als String-Konstante, String-Ausdruck oder String-Variable übergeben werden.

Soll diese Infozeile genutzt werden, wird sie **vor** dem entsprechenden OPENW durch INFOW angelegt. Wurde ein Fenster ohne Infozeile geöffnet, so muß es erst wieder geschlossen werden, um INFOW dort einsetzen zu können. Soll die Infozeile gelöscht werden, muß ebenfalls vorher das Fenster geschlossen werden, um mit INFOW einen Null-String ("") übergeben zu können (anschließend wieder OPENW).

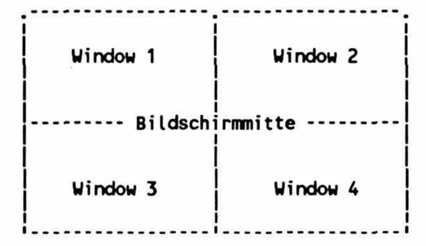
Am Ende des Kapitels finden Sie ein Beispiel für alle Window-Befehle.

OPENW { O W } Fenster öffnen

OPENW Nummer [,Xcenter,Ycenter]
OPENW #Nummer,Xpos,Ypos,Breite,Höhe,Attribute **(nur V3.0)**

OPENW 'Nummer' öffnet bzw. aktiviert das Window, dessen Nummer (0-4) übergeben wird..

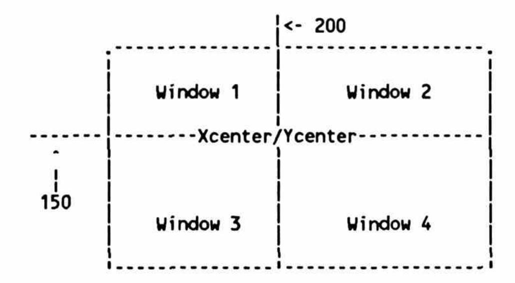
Die Fenster sind dabei zuerst ohne Überlappungen folgendermaßen angeordnet:



Wird die optionale Koordinatenangabe Xcenter,Ycenter eingesetzt, so berühren sich die Fenster an dem Bildschirmpunkt, der durch Xcenter/Ycenter bestimmt wurde. Dieser Schnittpunkt kann beim Öffnen oder Aktivieren jedes Windows durch neue Xcenter/Ycenter verändert werden. Er ist dabei unabhängig von einem evtl. durch OPENW 0 vorgegebenen Grafik-Nullpunkt (siehe unten).

```
      OPENW 1,200,150
oder  OPENW 2,200,150
oder  OPENW 3,200,150
oder  OPENW 4,200,150
```

hätte folgende Wirkung:



Die Windows werden dabei nicht in ihrer eigentlichen Lage verändert, sondern nur in ihrer Größe. Ihre relative Stellung zueinander bleibt erhalten.

Die zweite Syntax-Variante gilt nur für V3.0 und ermöglicht es, über einen einzigen Befehl das gesamte Fenster zu definieren. Dadurch kann die folgende Vorgehensweise aus den V2.xx-Versionen ersetzt werden:

```
Dpoke Windtab+2+(Nummer-1)*12,Attribute
Dpoke Windtab+4+(Nummer-1)*12,Xpos
Dpoke Windtab+6+(Nummer-1)*12,Ypos
Dpoke Windtab+8+(Nummer-1)*12,Breite
Dpoke Windtab+10+(Nummer-1)*12,Höhe
Openw 1
```

Die hiermit übergebenen Koordinaten beziehen sich dabei absolut auf die linke obere des jeweiligen Fensters und lassen die übrigen Fenster unverändert.

Die Bedeutung der einzelnen Parameter finden Sie unter WINDTAB beschrieben.

Beim ersten Öffnen eines Fensters mit den Nummern 1 - 4 wird automatisch ein nicht sichtbares Fenster mit dem Nummer 0 geöffnet, das den Menüzeilenbereich am oberen Bildrand ausspart.

Window 0 ist kein Fenster, das Sie in der Größe definieren können. Es bildet nur den Hintergrund zu einem evtl. vorhandenen Pull-Down-Menü. Würde das Window 0 nicht geöffnet und Sie benutzen keine weiteren Fenster, so würde die Menüleiste von Text- oder Grafik-Ausgaben überschrieben werden. Öffnen Sie dagegen Window 0, wird der Bildschirm am oberen Rand einfach um 18 Pixel abgeschnitten. In diesem 18 Pixel breiten Streifen hat jetzt ggfs. die Kopfzeile eines Pull-Down-Menüs Platz und alle Ausgabe-Befehle nehmen nun Pixel X=0/Y=19 des gesamten Bildschirms als Ursprung an.

Bei Programmende oder im Falle, daß der gesamte TOS-Bildschirm aktiviert werden soll, ist dieses Window vorher mit CLOSEW 0 zu schließen. Sind alle anderen Fenster geschlossen, kann durch OPENW 0,X_pos,Y_pos der Bildschirm-Nullpunkt beliebig bestimmt werden (siehe CLIP). Auf diesen Nullpunkt beziehen sich dann alle weiteren GEM/VDI-Ausgaben (TEXT, CIRCLE, BOX etc.).

Am Ende des Kapitels finden Sie ein Beispiel für alle Window-Befehle.

TITLEW { TIT } Fenster-Titelzeile bestimmen

In V3.0: { TI }

TITLEW Nummer,"Text" TITLEW [#]Nummer,"Text" (nur V3.0)

Das Fenster mit der angegebenen Nummer erhält in dem grauen MOVE-Balken die Überschrift Text.

Bis auf zwei Abweichungen gelten hier dieselben Erläuterungen wie zu INFOW. Die Abweichungen zu INFOW sind:

```
Text = "  "  => 2 Space      = Keine Überschrift
Text = ""    => Null-String = Fenster ist unbeweglich.
```

Version 3.0

TOPW {TO} — Fenster aktivieren

TOPW #Nummer

Dieser V3.0-Befehl aktiviert das Fenster mit der Nummer #Nummer. Das Fenster muß vorher durch OPENW geöffnet worden sein. Liegt es vor TOPW-Aufruf unter einem anderen Fenster, wird es "nach oben" gelegt und aktiviert. Im Gegensatz zu OPENW wird dabei intern keine Ab- und Anmeldung vorgenommen, was die Absturzgefahr bei "leichtsinniger" Fenster-Programmierung erheblich verringert.

Version 3.0

W_HAND() — GEM-Window-Handle ermitteln

Var=W_HAND(Nummer)

Liefert in V3.0 das Original-GEM-Handle des durch die GFA-Fensternummer Nummer angegebenen Windows. W_HAND() ist die Umkehrfunktion zu W_INDEX().

Version 3.0

W_INDEX() — GFA-Window-Nummer ermitteln

Var=W_INDEX(Handle)

Liefert in V3.0 die GFA-Fensternummer des durch das Original-GEM-Handle Handle angegebenen Windows. W_INDEX() ist die Umkehrfunktion zu W_HAND().

WINDTAB Fenster-Verwaltungstabelle

Var=DPEEK(WINDTAB)
DPOKE WINDTAB,Wert

Die ist die Start-Adresse eines Speicherbereichs, aus welchem Window-Verwaltungsdaten gelesen, bzw. in den solche nach Bedarf auch geschrieben werden können.

WINDTAB-Tabelle (je ein Word):

```
Offset:     Bedeutung:

 +0         GEM-Handle für Window 1
 +2         Attribute für Window 1 (siehe 'Attribut')
 +4         X-Koordinate für Window 1
 +6         Y-Koordinate für Window 1
 +8         Breite Window 1
+10         Höhe Window 1
+12         -.
bis          |--  Entsprechende Angaben für Window 2
+22         -'
+24         -.
bis          |--  Entsprechende Angaben für Window 3
+34         -'
+36         -.
bis          |--  Entsprechende Angaben für Window 4
+46         -'
+48         Reserviert (immer -1)
+50         Reserviert (immer  0)
                                          Hires Midres Lowres
+52         GEM-Bildschirm-X-Koordinate    (0)    (0)    (0)
+54         GEM-Bildschirm-Y-Koordinate   (19)   (10)   (10)
+56         GEM-Bildschirmbreite         (639)  (639)  (319)
+58         GEM-Bildschirmhöhe           (381)  (190)  (190)
+60         X-Schnittpunkt der vier Fenster     (siehe OPENW)
+62         Y-Schnittpunkt der vier Fenster     (siehe OPENW)
+64         X-Nullpunkt für Grafikbefehle
+66         Y-Nullpunkt für Grafikbefehle
```

Durch die letzten beiden Elemente der Tabelle hat man eine weitere Möglichkeit in V2.xx den Grafik-Nullpunkt selbst zu bestimmen (siehe CLIP und OPENW).

Bestimmung der aktiven Randelemente:

In V3.0:

```
WINDTAB(Nummer,1)=Attribut
```

In V2.xx:

```
DPOKE WINDTAB+2+(Nummer-1)*12,Attribut
```

```
                           ^            ^
      .--------------'            '----.
      !                                 !
      !                                 !
Nummer    = GFA-Window-Nummer (1-4).    !
                                        !
                                        !
      .---------------------------------'
      !
      !
Attribut ist ein 12-Bit-Wert, dessen gesetzte Bits
die Komponenten des Fensters bestimmen:
Bit 0   =  NAME des Fensters (Titelzeile)
Bit 1   =  CLOSE-Feld links oben
Bit 2   =  FULL-Feld rechts oben
Bit 3   =  MOVE-Balken oben (grauer Balken)
Bit 4   =  INFO-Zeile unter dem MOVE-Balken
Bit 5   =  SIZE-Feld rechts unten
Bit 6   =  UPARROW rechts (Aufwärtspfeil)
Bit 7   =  DNARROW rechts (Abwärtspfeil)
Bit 8   =  VSLIDE-Balken rechts (Vertikalschieber)
Bit 9   =  LFARROW unten (Linkspfeil)
Bit 10  =  RTARROW unten (Rechtspfeil)
Bit 11  =  HSLIDE-Balken unten (Horizontalschieber)
z.B.:

Dpoke Windtab+2,&X111111111111
    ==> Es werden alle Komponenten von
        Window 1 aktiviert.

Dpoke Windtab+2,&X001001001001
    ==> Es wird nur NAME,MOVER,PFEIL hoch
        und PFEIL links von Window 1 gesetzt.
usw.
```

Wurde bei Programmstart mit TITLEW eine Kopfzeile, bzw. mit INFOW eine Infozeile eingerichtet, bleibt sie auch aktiv, wenn hier die entsprechenden Bits **nicht** gesetzt werden.

Version 3.0

WINDTAB() — Fenster-Verwaltungstabelle als Array

Var=WINDTAB(Nummer,Index)
WINDTAB(Nummer,Index)=Wert

Dies ist eine Funktion, mit welcher die WINDTAB-Tabelle bis zum 46. Element auch über einen zweidimensionalen Index angesprochen werden kann.

Mit Nummer gibt man im ersten Parameter die GFA-Nummer des gewünschten Windows an (0 - 4). Der zweite Parameter Index steht für das Element der Tabelle, das ausgelesen oder verändert werden soll:

0	Window-Nummer.
1	Window-Attribute.
2	Window-X-Koordinate.
3	Window-Y-Koordinate.
4	Äußere Window-Breite.
5	Äußere Window-Höhe.

Beispiel:

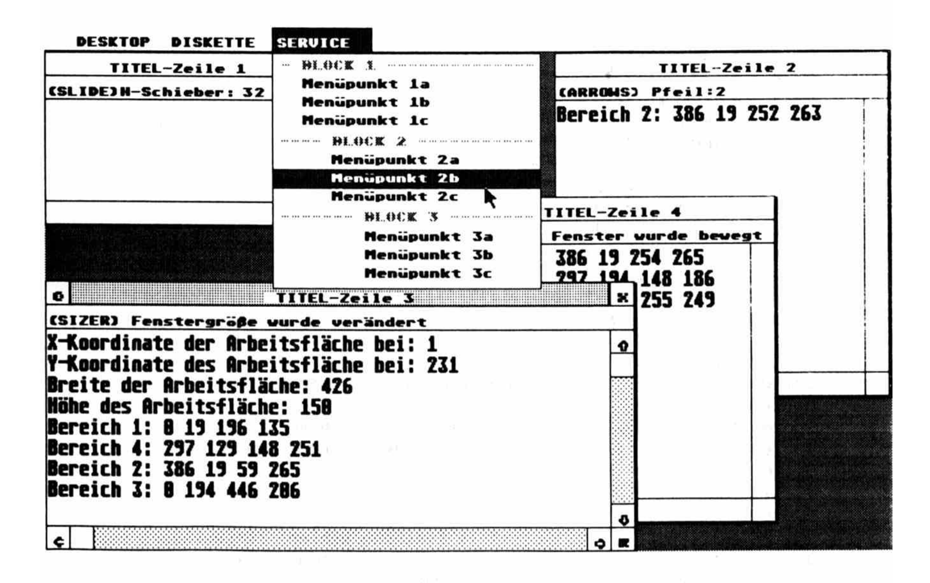
```
Openw #1,100,100,100,80,&HFFF  ! Window öffnen
Pause 50                       ! Kurz anschauen
Closew 1                       ! Window schließen
Windtab(1,1)=0                 ! Neue Attribute
Windtab(1,2)=10                ! Neue X-Koordinate
Windtab(1,3)=20                ! Neue Y-Koordinate
Windtab(1,4)=200               ! Neue Breite
Windtab(1,5)=175               ! Neue Höhe
Openw 1                        ! Fenster wieder öffnen
Clearw 1                       ! Fenster säubern
Pause 50                       ! Kurz anschauen
Closew 1                       ! Fenster schließen
Closew 0                       ! GEM-Fenster schließen
```

Beispiel zur Menü-, Window- und Ereignis-Verwaltung

Wer sich schon mal mit GEM-Programmierung befaßt hat, wird annähernd ermessen können, was es heißt, BASIC-Befehle zu entwickeln, die die Erstellung von Pull-Down-Menüs und Fenstern dermaßen vereinfachen, wie es mit diesem BASIC möglich ist.

Als Grundlage zur Erläuterung dieser Befehle soll uns ein Programm dienen, das mit einfachen Mitteln ein Pull-Down-Menü und die Verwendung von bis zu vier verschiedenen Fenster ermöglicht. Der Begriff "einfach" ist in diesem Fall relativ zu betrachten. Der Einsatz dieser Befehle birgt schon einige Klippen, die erst einmal, und das gilt speziell für Einsteiger, vorsichtig zu umgehen sind.

Dieses Programm ist so ausgelegt, daß es als Grundkonzept für Window- und Menüprogrammierung verwendet werden kann. Experimentieren Sie damit und beachten Sie die verschiedenen Effekte im Menü und in den Windows, wenn Sie z.B. Menüpunkte anwählen oder die Randelemente der Fenster anklicken.




```
On break gosub Ende         ! Break abfangen
'     Damit alle Fenster in jedem Fall ordnungsgemäß geschlossen
'     werden, bzw. das Menü ausgeschaltet wird, ist sicherzustellen,
'     daß auch bei der Break-Funktion <Control><Shift><Alternate>
'     zur Abschluß-Prozedur verzweigt wird.
'     Werden diese Abschlußarbeiten nicht ausgeführt, kann das zu
'     Abstürzen führen.
Dim M.punkt$(32)            ! Menü-Array dimensionieren
'     Es muß ein Text-Array angelegt werden, das die verschiedenen
'     Menütexte aufzunehmen hat. Dabei ist auf eine ausreichende
'     Dimensionierung zu achten. Es werden hier alle übergebenen
'     Strings gerechnet (also auch Leer-Strings).
Restore M.datas             ! Menü-DATA-Zeiger setzen
'     Die einzelnen Menüpunkt-Texte werden hier in DATA-Zeilen
'     abgelegt, aus denen Sie nun mit der folgenden Schleife
'     ausgelesen werden.
For I=0 To 31               ! 31 Menü-Texte
  Read M.punkt$(I)          ! ...lesen
  '   Durch diese Schleife werden die einzelnen Data-Strings dem
  '   Menütextfeld zugeordnet.
  Exit if M.punkt$(I)="~~~~" ! Exit, wenn Endmarkierung erreicht
Next I
M.punkt$(I)=""                  ! Letztes Menü abschließen
M.punkt$(I+1)=""                ! Gesamtes Pull-Down-Menü abschließen
M.datas:
Data DESKTOP
Data  Information
Data --------------------
Data  , , , , , ,""
Data DISKETTE
Data   Datei laden
Data   Datei speichern
Data   Datei einrichten
Data   Datei löschen
```

```
Data ----------------, Quit,""
Data SERVICE
Data - BLOCK 1 ---------------
Data   Menüpunkt 1a
Data   Menüpunkt 1b
Data   Menüpunkt 1c
Data ---- BLOCK 2 ------------
Data      Menüpunkt 2a
Data      Menüpunkt 2b
Data      Menüpunkt 2c
Data ------- BLOCK 3 ---------
Data         Menüpunkt 3a
Data         Menüpunkt 3b
Data         Menüpunkt 3c
Data ~~~~
Menu M.punkt$()            ! Menü installieren
Openw 0                    ! GEM-Haupt-Window öffnen
Dpoke Contrl+2,0           ! -----.
Dpoke Contrl+6,1                  !  Textart wählen
Dpoke Contrl+12,1                 !- siehe Prozedur Ttype
Dpoke Intin,1                     !  unter DEFTEXT
Vdisys 106                 ! -----'
Dpoke Contrl+2,0           ! -----.
Dpoke Contrl+6,1                  ! Texthöhe wählen
Dpoke Contrl+12,1                 !- siehe Prozedur Tsize
Dpoke Intin,9                     ! unter DEFTEXT
Vdisys 107                 ! -----'
Deftext ,1,0,13            ! PRINT-Texteinstellung
'    Mit dieser DEFTEXT-Einstellung können auch die PRINT-Ausgaben
'    im Fenster modifiziert werden. Wird die Schrifthöhe verändert,
'    verändern sich entsprechend auch die Cursor-Positionen (siehe
'    PRINT AT(x,y) bzw. CRSCOL/CRSLIN). Dies ist auch bei Window 0
'    möglich, ohne das weitere Fenster geöffnet werden. Die Ausgabe
'    von Text geht dann allerdings erheblich langsamer vor sich, da
'    dann auch PRINT-Texte über das VDI ausgegeben werden und nicht
'    mehr über das TOS.
Xt%=2-Sgn(Xbios(4))        ! X-Auflösungsteiler
Yt%=Min(2,3-Xbios(4))      ! Y-Auflösungsteiler
Deffill ,2,4               ! DEFFILL grau
Pbox 0,0,639/Xt%,380/Yt%   ! Desktop zeichnen
For I=1 To 4               ! 4 Fenster
  Dpoke Windtab+2+(I-1)*12,&X111111111110  ! definieren
  '   Hier werden die aktiven Window-Randelemente festgelegt.
  '   Die Bedeutung der Bits erfahren Sie unter WINDTAB.
  Infow #I," INFO-Zeile "+Str$(I)     ! Info-Zeile setzen
  Titlew #I," TITEL-Zeile "+Str$(I)   ! Titel-Zeile setzen
  Openw I                  ! Fenster öffnen
  Clearw I                 ! Fenster klar
Next I
Count=4                    ! Fenster-Zähler setzen
'    Diese Zählvariable hat den Zweck, festzustellen, ob alle
'    geschlossen wurden.
On menu gosub M.deal       ! Bei Pull-Down-Menü-Ereignis verzweigen
On menu message gosub M.essage ! Bei Message-Puffer-Ereignis   "
On menu key gosub K.ey     ! Bei Tastatur-Ereignis             "
On menu ibox 1,10,10,100,100 Gosub I.box ! Bei Ibox-Ereignis   "
On menu obox 2,10,10,100,100 Gosub O.box ! Bei Obox-Ereignis   "
Do                         ! Endlos-Schleife
```

```
  '   Dies ist die Hauptschleife des Programms. Innerhalb
  '   dieser Schleife wird nun je nach Bedarf das eigentliche
  '   Programm installiert.
  On menu                   ! Ereignisse feststellen
  Mouse X,Y,K               ! Maus-Status abfragen
  Xr=Dpeek(Windtab+8+(Menu(4)-1)*12)-20/Xt%  ! Window-Breite holen
  Yr=Dpeek(Windtab+10+(Menu(4)-1)*12)-60/Yt% ! Window-Höhe holen
  If K=1 And X=>0 And Y=>0 And X<Xr And Y<Yr
    ' Linker Mausklick innerhalb des aktuellen Windows?
    Print X'Y               ! Koordinaten ausgeben
  Endif
Loop
Procedure M.essage          ! Ziel-Prozedur für Message-Ereignisse
  If Menu(1)=21             ! Window aktualisieren (angeklickt)?
    Infow #Menu(4),"(TOPPER) Window "+Str$(Menu(4))+" gewählt"
    Openw Menu(4)           ! Gewünschtes Fenster öffnen
    Clearw Menu(4)          ! ...und säubern
  Endif
  If Menu(1)=22             ! Window schließen (CLOSE-Feld)?
    Closew Menu(4)          ! Aktuelles Window schließen
    Dec Count               ! Fensterzähler minus 1
    If Count=0              ! Alle Fenster zu?
      Alert 2,"Programm-Ende ?",1,"OKAY|NEIN",D%
      If D%=1               ! Abbruch?
        @Ende               ! Dann Ende
      Else
        Alert 1,"Fenster öffnen:|Tasten 1 - 4",1,"OKAY",D%
      Endif
    Endif
  Endif
  If Menu(1)=23             ! Window auf Maximal-Maß (FULL-Feld)?
    Closew Menu(4)          ! Aktuelles Window schließen
    Dpoke Windtab+4+(Menu(4)-1)*12,0        ! Neue X-Koordinate
    Dpoke Windtab+6+(Menu(4)-1)*12,19/Yt%   ! Neue Y-Koordinate
    Dpoke Windtab+8+(Menu(4)-1)*12,639/Xt%  ! Neue Breite
    Dpoke Windtab+10+(Menu(4)-1)*12,380/Yt% ! Neue Höhe
    '  In dieser Routine würde eigentlich allein der Befehl
    '  FULLW Handle% ausreichen. Dieser Befehl arbeitet
    '  leider nicht immer korrekt.
    '  Er läßt sich jedoch leicht simulieren, indem man das
    '  betreffende Fenster schließt, in die WINDTAB-Elemente,
    '  welche die Fenster-Koordinaten enthalten, die gewünschten
    '  Koordinaten einträgt und dann das Fenster wieder öffnet.
    Infow #Menu(4),"(FULLER) Volle Größe !" ! neue Info-Zeile
    Openw Menu(4)           ! Fenster wieder öffnen
    Clearw Menu(4)          ! ...und säubern
  Endif
  If Menu(1)=24             ! Window-Inhalt scrollen?
    '                       ! (siehe unter MENU(1) -> 24)
    Infow #Menu(4),"(ARROWS) Pfeil:"+Str$(Menu(5))
  Endif
  If Menu(1)=25             ! Window-Inhalt horizontal verschieben?
    '                       ! (siehe unter MENU(1) -> 25)
    Infow #Menu(4),"(SLIDE)H-Schieber : "+Str$(Menu(5))
  Endif
  If Menu(1)=26             ! Window-Inhalt vertikal verschieben?
    '                       ! (siehe unter MENU(1) -> 26)
    Infow #Menu(4),"(SLIDE)V-Schieber : "+Str$(Menu(5))
  Endif
```

```
  If Menu(1)=27             ! Window-Größe ändern (SIZE-Feld)?
    Closew Menu(4)          ! Aktuelles Fenster schließen
    Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7) ! Neue Breite eintragen
    Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)! Neue Höhe eintragen
    '  Falls das Größenfeld rechts unten gewählt wurde, werden
    '  den ensprechenden Elementen der WINDTAB-Tabelle die
    '  neuen Window-Maße übergeben.
    Infow #Menu(4),"(SIZER) Fenstergröße verändert"
    Openw Menu(4)           ! Fenster wieder öffnen
    Clearw Menu(4)          ! Fenster säubern
    @Get.wholearea          ! Koordinaten und Maße der inneren
    '                       ! Arbeitsfläche holen
    Print "X-Koordinate der Arbeitsfläche bei: ";Workx
    Print "Y-Koordinate des Arbeitsfläche bei: ";Worky
    Print "Breite der Arbeitsfläche: ";Workb
    Print "Höhe des Arbeitsfläche: ";Workh
  Endif
  If Menu(1)=28             ! Window verlegen (MOVE-Balken)?
    Closew Menu(4)          ! Aktuelles Window schließen
    Dpoke Windtab+4+(Menu(4)-1)*12,Menu(5) ! Neue X-Koordinate
    Dpoke Windtab+6+(Menu(4)-1)*12,Menu(6) ! Neue Y-Kooridnate
    Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7) ! Neue Breite
    Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)! Neue Höhe
    Infow #Menu(4),"(MOVER) Fenster wurde bewegt"
    Openw Menu(4)           ! Window wieder öffnen
    Clearw Menu(4)          ! Window säubern
    '  Das Fenster wurde bewegt. Die Windtab-Tabelle und die
    '  Info-Zeile werden aktualisiert und das Fenster mit den
    '  neuen Koordinaten gezeichnet.
  Endif
Return
Procedure M.deal            ! Ziel-Prozedur bei Menü-Ereignissen
  Menu off                  ! Titel-Zeile wieder weiß schalten
  Cls                       ! Aktuelles Fenster säubern
  Print "Menu 0 :";Menu(0);" Menu 1 :";Menu(1)   !--.
  Print "Menu 2 :";Menu(2)," Menu 3 :";Menu(3)     !
  Print "Menu 4 :";Menu(4);" Menu 5 :";Menu(5)     !
  Print "Menu 6 :";Menu(6)," Menu 7 :";Menu(7)     !  Gesamten
  Print "Menu 8 :";Menu(8);" Menu 9 :";Menu(9)     !- Message-
  Print "Menu 10:";Menu(10)," Menu 11:";Menu(11)   !  Puffer
  Print "Menu 12:";Menu(12)," Menu 13:";Menu(13)   !  ausgeben
  Print "Menu 14:";Menu(14)," Menu 15:";Menu(15)   !
  Print "Menu -1 (Objektbaumadresse) :";Menu(-1) !--'
  Mp%=Menu(0)               ! Index des gewählten Menüpunktes
  '   In Menu(0) wird generell der Index des angewählten
  '   Menüpunktes abgelegt. Dieser Index ist identisch mit
  '   dem Index des entsprechenden Menüpunkt-Textelementes
  '   im Menütext-Feld.
  '
  '   Die folgenden Abfragen beziehen sich auf das jeweils
  '   stattgefundene Ereignis und verwenden zur Feststellung
  '   den im entsprechenden Menütext-Element enthaltenen Text.
  '   Der zur Abfrage verwendete String muß dabei mit dem
  '   Menütext-String absolut (inkl. Leerzeichen) identisch sein.
  If M.punkt$(Mp%)=" Information"
    Al.str$="Geschlossene Fenster können|über die Ziffern"
    Al.str$=Al.str$+"tasten 1 - 4|wieder geöffnet werden !"
    Alert 1,Al.str$,1,"OKAY",Dummy
  Endif
```

```
Menu Mp%,2              ! Menüpunkt grau (inaktivieren)
Menu Mp%,1              ! Menüpunkt "abhaken" (markieren)
Menu Mp.mem%,3          ! Vorherigen Menüpunkt (aktivieren)
Menu Mp.mem%,0          ! Markierung d. vorherigen Menüpunkts
'                       ! löschen
Mp.mem%=Mp%             ! Index des aktuellen Menüpunkts merken
If M.punkt$(Mp%)=" Datei laden"
  ' ** Lade-Routine **
Endif
If M.punkt$(Mp%)=" Datei speichern"
  ' ** Speicher-Routine **
Endif
If M.punkt$(Mp%)=" Datei einrichten"
  ' ** Init-Routine **
Endif
If M.punkt$(Mp%)=" Datei löschen"
  ' ** Kill-Routine **
Endif
If M.punkt$(Mp%)=" Quit"
  ' ** Quit-Routine **
  @Ende
Endif
If M.punkt$(Mp%)="  Menüpunkt 1a"
  ' ** Menü-Routine 1 **
  M.punkt$(Mp%)="  1a umgekehrt"   ! Neuen Text zuordnen
  Menu M.punkt$()                  ! Menü neu initialisieren
  ' Es ist auch eine nachträgliche Änderung des Menütextes
  ' möglich. Dies wird interessant, wenn ein angeklickter
  ' Menüpunkt nach Ausführung der ihm zugeordneten Funktion
  ' z.B. in die gegenteilige Funktion verwandeln soll (z.B.
  ' von "schwarz" zu "weiß" - und umgekehrt).
  ' Dazu muß das Element des Menütext-Feldes mit dem Index
  ' des gewählten Menüpunktes verändert werden. Den Index
  ' des Punktes finden Sie in MENU(0).
  ' Beim Auswechseln des Textes ist darauf zu achten, daß
  ' der neue Menütext nicht länger ist als der alte (Absturz).
  ' Ist der Text ausgewechselt, muß mit MENU'Array$' das Menü
  ' neu initialisiert werden. Durch eine Operation dieser Art
  ' mit allen Menüpunkt-Texten kann man folglich während des
  ' Programms auch das komplette Menü auswechseln, womit also
  ' auch der Einsatz mehrerer Menüs gleichzeitig möglich ist.
  ' Wenn mehr Menüpunkte existieren, als mit einem Menü auf
  ' dem Bildschirm grafisch darstellbar sind, kann man so
  ' mehrere Menüs aufbauen, die sich dann gegenseitig aufrufen
  ' bzw. sich miteinander ergänzen.
Else
  If M.punkt$(Mp%)="  1a umgekehrt"
    ' ** Alternative zu Menü-Routine 1 **
    M.punkt$(Mp%)="  Menüpunkt 1a"
    Menu M.punkt$()
  Endif
Endif
If M.punkt$(Mp%)="  Menüpunkt 1b"
  ' ** Menü-Routine 2 **
Endif
If M.punkt$(Mp%)="  Menüpunkt 1c"
  ' ** Menü-Routine 3 **
Endif
If M.punkt$(Mp%)="    Menüpunkt 2a"
```

```
    ' ** Menü-Routine 4 **
  Endif
  If M.punkt$(Mp%)="    Menüpunkt 2b"
    ' ** Menü-Routine 5 **
  Endif
  If M.punkt$(Mp%)="    Menüpunkt 2c"
    ' ** Menü-Routine 6 **
  Endif
  If M.punkt$(Mp%)="        Menüpunkt 3a"
    ' ** Menü-Routine 7 **
  Endif
  If M.punkt$(Mp%)="        Menüpunkt 3b"
    ' ** Menü-Routine 8 **
  Endif
  If M.punkt$(Mp%)="        Menüpunkt 3c"
    ' ** Menü-Routine 9 **
  Endif
Return
Procedure K.ey             ! Ziel-Prozedur bei Tastatur-Ereignissen
  Print "Scan- + ASCII-Code : ";Menu(14) ! Code anzeigen
  Taste$=Right$(Mki$(Menu(14)))          ! ASCII-Zeichen
  Print "Scan-Code  : "'Peek(Varptr(Taste$))  ! Scan-Code ausgeben
  Print "ASCII-Code : "'Peek(Varptr(Taste$)+1)! ASCII-Code    "
  If Taste$=>"1" And Taste$=<"4" ! 1 bis 4 gedrückt?
    Openw Val(Right$(Taste$))    ! Entsprechendes Window öffnen
    Clearw Val(Right$(Taste$))   ! Entsprechendes Window säubern
    Inc Count                    ! Fensterzähler plus 1
  Endif
Return
Procedure I.box            ! Ziel-Prozedur bei Ibox-Ereignissen
  If Flag=0                ! Ibox-Flag nicht gesetzt?
    Print "Mauszeiger befindet sich in der Box."
    Flag=1                 ! Ibox-Flag setzen
  Endif
Return
Procedure O.box            ! Ziel-Prozedur bei Obox-Ereignissen
  If Flag=1                ! Maus war in der Box?
    Print "Mauszeiger befindet sich außerhalb der Box."
    Flag=0                 ! Ibox-Flag löschen
  Endif
Return
Procedure Get.wholearea
  '   In dieser Prozedur wird die Größe und Position der
  '   Arbeitsfläche des jeweiligen Fensters ermittelt. Dies
  '   ist eine AES-Funktion, die Sie nach Bedarf verwenden
  '   können.
  '   Die Ermittlung der Arbeitsflächen-Koordinaten und
  '   -Maße ist zur Restauration von Fensterinhalten
  '   (Window-Redraw) unumgänglich.
  Dpoke Gintin,Menu(4)
  Dpoke Gintin+2,4
  Gemsys 104
  Workx=Dpeek(Gintout+2)
  Worky=Dpeek(Gintout+4)
  Workb=Dpeek(Gintout+6)
  Workh=Dpeek(Gintout+8)
Return
Procedure Ende             ! Break-Abfang-Prozedur
  Closew 4                 !--.
```

```
  Closew 3                ! Alle Fenster
  Closew 2                !- schließen
  Closew 1                !
  Closew 0              !--'
  Menu kill             ! Menü ausschalten
  Cls                   ! Bildschirm löschen
  Edit                  ! Zurück zum Editor
Return
```

15. Menüprogrammierung mit BASIC-Befehlen

MENU menü — Menüpunkt-Attribute bestimmen

MENU Menüpunkt,Attribut

Mit diesem Befehl kann bestimmt werden, ob ein Menüpunkt grau (inaktiv) dargestellt werden soll oder schwarz (aktiv). Außerdem können vor einem Menüpunkt-Text Häkchen (Checkmarks) gesetzt werden. Diese kennzeichnen üblicherweise, daß die Funktion, die durch den gewählten Menüpunkt repräsentiert wird z.Zt. aktiv ist.

Dazu wird dem Befehl in Menüpunkt der Index des zu markierenden bzw. des zu invertierenden Menüpunktes (siehe MENU(0)), sowie nach einem Komma das gewünschte Attribut übergeben.

Attribut:

1	**Checkmark löschen.**
1	**Checkmark setzen (abhaken).**
2	**Menüpunkt deaktivieren (hell).**
3	**Menüpunkt aktivieren.**

Bei MENU Menütext$ sind für den Fall, daß Checkmarks (Häkchen) gesetzt werden sollen, vor der jeweiligen Menüpunkt-Bezeichnung zwei Leerzeichen vorzusehen.

Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

MENU menütext$() — Pull-Down-Menü erstellen

MENU Array$()

Mit diesem Befehl wird dem Interpreter gesagt, wo er im Speicher die einzelnen Menüeinträge findet. Sie übergeben ihm dazu in Klammern den Namen eines eindimensionalen String-Arrays, das die Menütexte enthält. Dieses Array muß soviele Elemente aufweisen, wie insgesamt an Menüpunkt-Texten zugewiesen werden sollen.

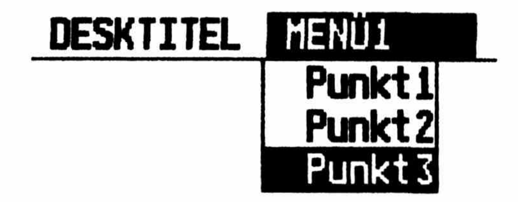
Hierbei ist zu beachten, daß das erste Menü (Desktop-Menü für Accessories) folgenden Aufbau hat:

1. String = Menütitel (evtl. Programmname).
2. String = Beliebige Überschrift (üblich: Programm-Info). Unter diesem Menüpunkt können Sie eine beliebige Programm-Funktion einordnen. Da es aber der einzige verwendbare Menüpunkt in diesem ersten Menü ist, wird es normalerweise für die Ausgabe einer allgemeinen Programm-Information (z.B. Copyright) verwendet.
3. String = Reihe von Minuszeichen (bzw. Bindestrichen). Die Anzahl der Striche bestimmt hier die Menübreite. Da in diesem Menü grundsätzlich die evtl. vorhandenen Desk-Accessories aufgeführt werden, sollten Sie hier die Länge des längsten Accessory-Titels berücksichtigen.
4. bis 9. String = Sechs Leer-Strings (mind. " "), die als Platzhalter für die Accessory-Titel dienen. Es dürfen hier keine Null-Strings ("") übergeben werden!
10. String = Null-String. Dieser Null-String gilt als Abschluß für jede einzelne Menüreihe.

An diesen ersten Menüaufbau werden nun die von Ihnen frei benennbaren weiteren Menüs angehängt. Diese sind so aufgebaut, daß sich an den Menütitel (der String der immer in der Menüleiste sichtbar bleibt) die einzelnen Menüpunkt-Bezeichnungen anschließen. Dabei muß auch hier, wie beim ersten Menü, ein Null-String ("") den Abschluß zu jedem einzelnen Menü bilden.

Nachdem alle Menüeinträge gelesen wurden, muß als Endmarkierung dem gesamten Menüaufbau noch ein Null-String als Endmarkierung angehängt werden. Wollen Sie, daß bei Anwahl eines Menüpunktes dieser mit einem Häkchen (Checkmark siehe MENU Menüpunkt,Attribut) versehen wird, sollten Sie vor dem Text des gewünschten Menüpunktes zwei Leerzeichen vorsehen, damit dieses Checkmark genügend Platz hat.

Außerdem ist es möglich, einen Menüpunkt-Text invers (grau = inaktiv) darzustellen. Dazu muß das erste Zeichen des Text-Strings ein Minuszeichen (bzw. Bindestrich: -) sein. Nachdem das Menü eingelesen wurde, wird es durch diesen Befehl auch gleichzeitig aktiviert.



Beispiel:

```
DIM Array$(30)
REPEAT
  READ Array$(I%)
```

```
  INC I%
UNTIL Array$(I%)="XXX"
Array$(I%)=""
DATA DESKTITEL,Menüpunkt
DATA --------------------------------
DATA Acc1,Acc2,Acc3,Acc4,Acc5,Acc6,""
DATA MENÜ1,  Punkt1,  Punkt2,  Punkt3,""
DATA MENÜ2,  Punkt1,  Punkt2,  Punkt3,""
DATA MENÜ3,  Punkt1,  Punkt2,  Punkt3,""
DATA MENÜ4,  Punkt1,  Punkt2,  Punkt3,""
DATA XXX
MENU Array$()
```

Als weiteres Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

Zu jedem besseren Programm gehört unvermeidlich, daß die evtl. vorhandenen Desktop-Accessories angewählt werden können. Dazu muß man nicht "unbedingt" reine GEM-Programme mit Windows, Menüs und allem Drum und Dran schreiben, sondern es reicht, das Pull-Down-Menü mit den Accessories nur dann zur Verfügung zu stellen, sobald es vom Anwender gewünscht wird.

Zu diesem Zweck folgt nun eine Prozedur, die diesen Vorgang selbstständig erledigt, wobei auch der Hintergrund nach Rückkehr zum Programm selbsttätig wieder restauriert wird. Sollten während der Warteschleife weitere ON MENU GOSUBs aktiv sein, ist darauf zu achten, daß wieder zur Warteschleife zurückgekehrt wird, da sonst das Menü und der Hintergrund nicht korrekt restauriert werden können.

```
Deffill ,2,2
Pbox 10,10,630,390
Print "Maustaste drücken"
Do
  Repeat
  Until Mousek
  @Access("ACCESSORY"," ZURÜCK ZUM PROGRAMM")
Loop
'
Procedure Access(Ac.titel$,Ac.exit$)
  ' Ac.titel$ = gewünschter Titel für das Accessory-Menü.
  ' Ac.exit$  = Text des ersten Menüpunkts im Accesory-Menü.
  '             Gilt hier als Exit-Menüpunkt!
  Local M.i%,M.bg$          ! Lokale Variablen
  Pause 6                   ! Kleine Klickpause
  Sget M.bg$                ! Hintergrund sichern
  For M.i%=Xbios(2)+160 To Xbios(2)+1520 Step 160
    Bmove Varptr(M.bg$),M.i%,Max(1,32000-(M.i%-Xbios(2)))
    ' Bildschirm-Inhalt "soft" abwärts scrollen
  Next M.i%
  Dim M_feld$(10)           ! Menütext-Feld einrichten
  M_feld$(0)=Ac.titel$      ! Accessory-Menütitel zuordnen
  M_feld$(1)=Ac.exit$       ! Ersten Menüpunkt zuordnen
```

```
    M_feld$(2)="-----------------------" ! Menübreite
    For M.i%=3 To 8          ! 6 mal
      M_feld$(M.i%)="  "     ! 2-Space-String übergeben
    Next M.i%
    M_feld$(9)=""            ! 1. Endmarkierung setzen
    M_feld$(10)=""           ! 2. Endmarkierung setzen
    Menu M_feld$()           ! Menü initialisieren
    On menu gosub Acc_menue_xxx ! Menü-Überwachung anmelden
    Do                       ! Endlos-Schleife
      On menu                ! Ereignisse überwachen
      Exit if M_feld$(Menu(0))=Ac.exit$ ! Exit, wenn
    Loop                     ! Exit-Menüpunkt gewählt wurde
    Menu kill                ! Menü ausschalten
    For M.i%=Xbios(2)+1520 To Xbios(2)+160 Step -160
      Bmove Varptr(M.bg$),M.i%,Max(1,32000-(M.i%-Xbios(2)))
      ' Bildschirm-Inhalt "soft" aufwärts scrollen
    Next M.i%
    Sput M.bg$               ! Screen restaurieren
    Erase M_feld$()          ! Menütext-Feld löschen
  Return
  Procedure Acc_menue_xxx    ! Ziel-Prozedur für ON MENU GOSUB
    Menu off                 ! Menütitel wieder weiß schalten
  Return
```

MENU KILL — Menüzeile löschen

MENU KILL

MENU KILL deaktiviert das Pull-Down-Menü. Es kann keine Auswahl mehr vorgenommen werden, obwohl der Menütext stehen bleibt. Auf den Inhalt des Menütext-Arrays hat dieser Befehl keinen Einfluß. Das deaktivierte Menü kann also jederzeit durch MENU menütext$ wieder installiert werden.

Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

MENU OFF — Menütitel invertieren

MENU OFF

Dieser Befehl sollte immer dann eingesetzt werden, wenn ein Menü geöffnet und ein Menüpunkt gewählt wurde. Dadurch wird der bei der Öffnung invertierte Menütitel wieder auf "schwarz auf weiß" geschaltet. Wurde kein Menüpunkt gewählt und außerhalb des offenen Menüs ins Window geklickt, so ist diese Maßnahme nicht nötig. Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

16. Ereignis-Überwachung mit BASIC-Befehlen

MENU(Index) Event-Puffer (Menü- und Fensterverwaltung)

Var=MENU(Index)

Hinter dieser Funktion verbirgt sich ein Vektor (Event-Buffer = Integerfeld), in welchem permanent verschiedene Daten zum aktuellen GEM-Multi-Ereignis eingetragen werden.

Index steht für das jeweils interessante Feldelement. MENU(1) bis MENU(8) ist auf den aktuellen Message-Puffer gelegt und MENU(9) bis MENU(15) auf das INTOUT-Array. In MENU(5) kann die Bedeutungen des Eintrags von Fall zu Fall variieren. Bei Events, die ein Fenster in Größe und Lage verändern, werden in MENU(5) bis MENU(8) die entsprechenden Window-Koordinaten abgelegt. Bei den MENU(1)-Ereignissen 20-29 kann grundsätzlich in MENU(4) das aktuelle GEM-Window-Handle ausgelesen werden. Die übrigen geänderten MENU-Einträge sind unten unter dem jeweiligen MENU(1)-Identifikator (siehe "dann:") verzeichnet.

Die Inhalte von MENU(Index):

MENU(-2) Startadresse des Message-Puffers.
MENU(-1) Adresse des aktuellen Menü-Objektbaumes.
MENU(0) Menütext-Index des gewählten Menüpunktes.
MENU(1) Kenn-Nummer des jeweiligen Ereignisses: Mögliche Einträge:

10 = WM_SELECTED: Pull-Down-Menü wurde angewählt.
Dann: MENU(0) = Menü-Array-Index:
MENU(4) = Menütitel-Objektindex.
MENU(5) = Menüpunkt-Objektindex.
20 = WM_REDRAW: Window-Bereich neu zeichnen.
Dann: MENU(5) bis MENU(8) = X, Y, Breite, Höhe
21 = WM_TOPPED: Ein Window soll aktiviert werden.
22 = WM_CLOSED: Schließfeld links oben wurde gewählt.
23 = WM_FULLED: Vollfeld rechts oben wurde gewählt.
24 = WM_ARROWED: Window-Inhalt soll gescrollt werden. Es wurde einer der vier Pfeile oder einer der beiden grauen Scroll-Balken angeklickt.
Dann: MENU(5) = Ereignis-Index:

0 = Ganze Seite hoch (Klick auf vert. Scroll-Balken oberhalb des Schiebers).
1 = Ganze Seite runter (Klick auf vert. Scroll-Balken unterhalb des Schiebers).
2 = Eine Zeile hoch (Klick auf Aufwärtspfeil).
3 = Eine Zeile runter (Klick auf Abwärtspfeil).
4 = Ganze Seite nach links
(Klick auf horiz. Scroll-Balken links des Schiebers).

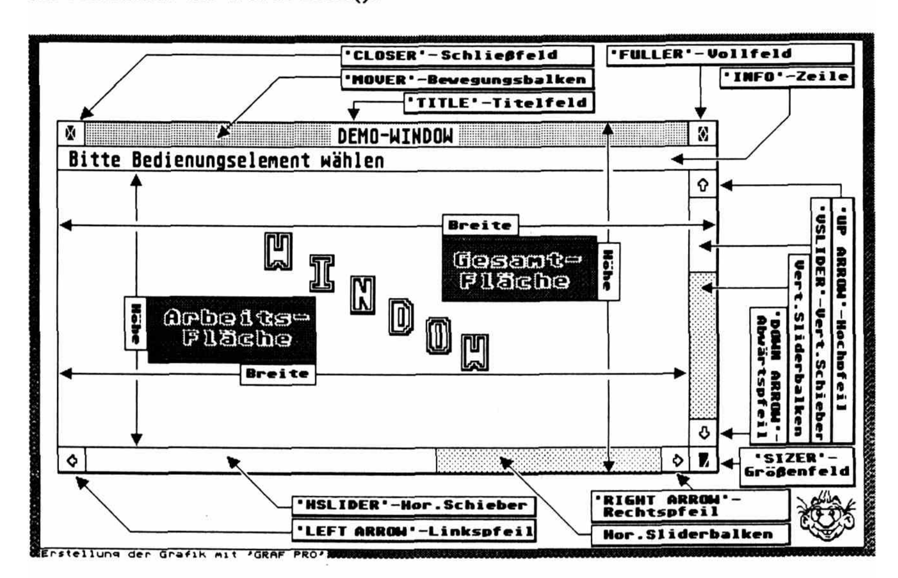
5 = Ganze Seite nach rechts
(Klick auf horiz. Scroll-Balken rechts des Schiebers).
6 = Ein Zeichen nach links (Klick auf Linkspfeil).
7 = Ein Zeichen nach rechts (Klick auf Rechtspfeil).

25 = WM_HSLID: Horizontaler Schieber wurde bewegt.
Dann: MENU(5) = Schieberposition (1 = links/1000 = rechts).
26 = WM_VSLID: Vertikaler Schieber wurde bewegt.
Dann: MENU(5) = Schieberposition (1 = oben/1000 = unten).
27 = WM_SIZED: Größenfeld rechts unten wurde gewählt.
Dann: MENU(5)/MENU(6) = Alte Fenster-X/Y-Koordinate.
MENU(7)/MENU(8) = Neue Fensterbreite/-höhe.
28 = WM_MOVED: Grauer Kopfbalken wurde gewählt.
Dann: MENU(5)/MENU(6) = Neue Fenster-X/Y-Koordinate.
MENU(7)/MENU(8) = Alte Fensterbreite/-höhe.
29 = WM_NEWTOP: Fenster wurde geschlossen. Das nächste vorhandene wird automatisch geöffnet.
30 = AC_OPEN: Accessory wurde angewählt (nur für das betroffene Accessory lesbar).
Dann: MENU(4) = Accessory-Identifikator:
(Index im Accessory-Menü)
Ist in MENU(4) die Kennziffer des lesenden Accesories enthalten, weiß es, daß es geöffnet werden soll.
31 = AC_CLOSE: Accessory wurde geschlossen (nur für das betroffene Accessory lesbar).
Dann: MENU(4) = Accessory-Identifikator
(Index im Accessory-Menü)
Ist in MENU(4) die Kennziffer des lesenden Accessories enthalten, weiß es, daß es geschlossen werden soll.

MENU(2) VDI-Handle der auslösenden Applikation.
MENU(3) Message-Länge in Bytes. Wenn 0, dann wurde nur MENU(1) - (8) geändert, sonst gibt MENU(3) die Länge der Message minus 16 Bytes für MENU(1)-(8) an.
MENU(4) GEM-Window-Handle oder Menütitel-Index.
MENU(5) X-Koordinate/Schieberposition/Randobjekt.
MENU(6) Y-Koordinate.
MENU(7) Breite.
MENU(8) Höhe.
MENU(9) Event-Flag:
0 = Kein Ereignis
Bit 0 gesetzt = Tastatur-Ereignis
Bit 1 gesetzt = Maustasten-Ereignis
Bit 2 gesetzt = I/Obox1-Ereignis
Bit 3 gesetzt = I/Obox2-Ereignis
Bit 4 gesetzt = Message-Ereignis
Bit 5 gesetzt = Timer-Ereignis
MENU(10) Maus-X-Koordinate bei Maus- und I/Obox-Events (rel. zur linken oberen Bildschirmecke).
MENU(11) Maus-Y-Koordinate bei Maus- und I/Obox-Events (rel. zur linken oberen Bildschirmecke).
MENU(12) Maustasten-Status bei Maus-Events (siehe MOUSEK).
MENU(13) Wechseltasten-Status bei Tastatur-Events:
Bit 0 = <Shift> rechts
Bit 1 = <Shift> links
Bit 2 = <Control>
Bit 3 = <Alternate>
MENU(14) Tastatur-Code (16-Bit-Wert) bei Tastatur-Events.
Highbyte (MENU(14) AND &HFF00 = Scan-Code
Lowbyte (MENU(14) AND &HFF = ASCII-Code

```
MENU(15)   Mausklickanzahl zum Zeitpunkt des Button-Events.
```

Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().



ON MENU — Verzweigung zur Ereignisfeststellung

ON MENU
ON MENU [Zeit] **(nur V2.02/V3.0)**

GFA-Multi-Event-Funktion: Stellt fest, ob ein AES-Multi-Ereignis (Mausbewegung, Tastatur, Window- und Fensteraktionen usw.) eingetreten ist. Dies sollte an häufig wiederkehrenden Programmstellen (innerhalb von Schleifen) geschehen, da der Mitteilungs-Puffer permanent erneuert wird. Wird nicht unverzüglich auf das eingetretene Ereignis reagiert und der Puffer ausgelesen, können neuere Puffer-Einträge die Mitteilung überschreiben und somit ein falsches Ergebnis liefern.

Wird der Befehl nicht eingesetzt, kann auch keine der ON MENU...GOSUB-Prozeduren angesprungen werden.

Ab V2.02 kann optional der Parameter Zeit eingesetzt werden. Dieser Wert gibt in 1000stel Sekunden an, nach welcher Zeitspanne ON MENU (siehe EVNT_MULTI) spätestens abgeschlossen werden soll

und die Kontrolle wieder an BASIC zurückgegeben wird. Bei ausreichender Zeitvorgabe hat so das AES Zeit, ggfs. das Loslassen des Mausknopfes zu registrieren. Dazu ist allerdings unbedingt erforderlich, ON MENU BUTTON GOSUB einzusetzen, da sonst der Zeitwert unberücksichtigt bleibt.

Vergessen Sie *nicht*, diesen Befehl auch innerhalb von internen Warteschleifen anzuführen (z.B. REPEAT...UNTIL MOUSEK), da sonst solange kein Ereignis bearbeitet werden kann, wie sich das Programm in dieser Warteschleife befindet.

Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

ON MENU GOSUB

Proc.-Bestimmung (Menü-Event)

ON MENU GOSUB Prozedurname

Prozedurname gibt die Prozedur an, zu welcher verzweigt werden soll, wenn ein Pull-Down-Menüpunkt (Menüeintrag) angeklickt wurde (sofern es kein Accessory war). Über MENU(0) kann dann dort der gewählte Menüpunkt ermittelt und dementsprechend reagiert werden.

Existiert Prozedurname nicht, wird die Menü-Überwachung abgeschaltet. Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

ON MENU BUTTON GOSUB

Proc.-Bestimmung (Mausknopf-Event)

ON MENU BUTTON Anz,Taste,Status GOSUB Prozedurname

Durch diesen Befehl kann eine Prozedur bestimmt werden, zu welcher verzweigt werden soll, sobald eine oder mehrere Maustasten ein- oder mehrmals gedrückt oder auch nicht gedrückt werden. Anz bestimmt die max. Anzahl an Mausklicks, die berücksichtigt werden sollen. Dies ist ein Maximalwert, der angibt, bis zu welcher Klickanzahl überhaupt reagiert werden soll.

Taste gibt an, auf welche Maustasten reagiert werden soll:

0	Keine
1	Linke
2	Rechte
3	Beide

Status bestimmt den Tasten-Status, der zum Ereignis führen soll:

0	Es muß keine Taste gedrückt worden sein.
1	Linke Taste muß gedrückt werden.
2	Rechte Taste muß gedrückt werden.
3	Beide Tasten müssen gedrückt werden.

Die tatsächlich ausgeführten Klicks können in Prozedurname aus MENU(15) gelesen werden. In MENU(15) steht allerdings fast immer der Wert 1, so daß Einzelklicks daraus kaum zu erkennen sind.

Allgemein werden sogenannte Doppelklicks eingesetzt, um irgendwelche erweiterten Maustastenfunktionen auszulösen. Auf dem Desktop wird z.B. ein Klick auf ein Programmsymbol als "aktiviert/gewählt" gewertet. Über das Desktop-Menü kann man nun bestimmte Auskünfte über diese Datei erhalten. Wird dagegen zweimal kurz hintereinander auf ein solches Symbol geklickt, so wird das Programm gestartet.

Ähnliches können Sie bei der Dateiauswahl in einer FILESELECT-Box beobachten. Ein Einfach-Klick führt dazu, daß der gewählte Dateiname in die Eintragszeile geschrieben wird, während ein Doppelklick dazu führt, daß die Box mit dem angeklickten Namen verlassen wird.

Bei der Wahl der Klick-Anzahl sollten Sie bedenken, daß ein "Dreifach-Klick" schon sehr schwer zu bewerkstelligen ist. Auch wenn Sie selbst derart "flinke Finger" haben, heißt das nicht, daß ein anderer Anwender Ihres Programms dieselben Klick-Kunststücke vollbringt.

Die Parameter dieses Befehls und ihre Auswirkungen sind nicht leicht zu durchschauen, da sich Taste und Status anscheinend widersprechen. Mir ist es bisher auch nicht gelungen, in allen Fällen klar vorher zu sagen, was die Parameterkonstellationen im einzelnen bewirken. Bei Verwendung ist es daher ratsam, die Wirkung jeweils zu testen.

Beispiel:

```
On menu button 3,2,2 Gosub Mausknopf ! Prozedur bestimmen
Print "Exit durch Dreifachklick der rechten Maustaste"
Do                  ! Endlos-Schleife
  On menu           ! Ereignis-Überwachung
```

```
Loop
Procedure Mausknopf  ! Ziel-Prozedur für Mausklick-Ereignisse
  Print At(10,10);Menu(15);" Klick(s) bei ";Menu(10);"/";Menu(11)
  If Menu(15)=3      ! Dreifachklick?
    Edit             ! Dann Programmende
  Endif
Return
```

ON MENU IBOX GOSUB

Procedure-Bestimmung (IBox/Maus-Event)

ON MENU OBOX GOSUB

Procedure-Bestimmung (OBox/Maus-Event)

ON MENU IBOX Id,X,Y,B,H GOSUB Prozedurname
ON MENU OBOX Id,X,Y,B,H GOSUB Prozedurname

Mit diesen beiden Befehlen können unabhängig voneinander vier Bildschirmbereiche definiert werden, die - vorausgesetzt, die Ereignis-Überwachung ist durch ON MENU aktiv - vom GEM ständig daraufhin überwacht werden, ob der Mauszeiger sie betritt (IBOX), bzw. verläßt (OBOX). Tritt eines dieser Ereignisse ein, wird zu Prozedurname verzweigt.

Id bestimmt, welcher der jeweils zwei unabhängigen Bereiche (Id 1 oder 2) überwacht werden soll.

X/Y geben die Koordinate der linken, oberen Ecke des jeweiligen Bereichs an. Die Koordinaten beziehen sich dabei nicht auf den Nullpunkt eines evtl. geöffneten Fensters, sondern auf den absoluten Nullpunkt in der linken, oberen Ecke des Bildschirms.

B/H geben die gewünschte Breite und Höhe des Rechtecks an. Existiert Prozedurname nicht, wird die Box-Überwachung abgeschaltet.

Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

ON MENU KEY GOSUB Proc.-Bestimmung (Tastatur-Event)

ON MENU KEY GOSUB Prozedurname

Die Tastatur wird überwacht und bei eingetretenem Ereignis (ON MENU nicht vergessen) zu Prozedurname verzweigt, wo man dann durch MENU(14) den Code der gedrückten Taste erfahren kann.

Existiert Prozedurname nicht, wird die Tastatur-Überwachung abgeschaltet. Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

ON MENU MESSAGE GOSUB Procedure-Bestimmung (Multi-Event)

ON MENU MESSAGE GOSUB Prozedurname

Überwacht die möglichen Multi-Events und verzweigt ggfs. zu Prozedurname. Über MENU(Index) (siehe dort) kann dann auf das jeweilige Ereignis reagiert werden.

Existiert die angegebene Prozedur nicht, wird die Event-Überwachung abgeschaltet. Als Beispiel beachten Sie das Abschluß-Beispiel der Window-Befehle im Anschluß an WINDTAB().

17. GDOS/VDI-Bibliothek

Die in diesem Kapitel beschriebenen Funktionen (außer VQT_EXTENT() und VQT_NAME()) stehen dem GFA-Programmierer nur zur Verfügung, wenn bei Systemstart im Auto-Ordner das Programm GDOS.PRG (Digital Research ab Version 1.1, besser: 1.8) enthalten war. Findet dieses Programm bei Ausführung - vor der eigentlichen System-Initialisierung - auf der Hauptebene der Boot-Station die (gültige) Datei ASSIGN.SYS vor, wird das GDOS in die GEM-Systemebene ge'linkt' und meldet sich (kurz vor Aufbau des Desktops) mit "GDOS V1.x resident".

Mit dieser GEM-Erweiterung ist - bei richtiger Anwendung - dann z.B. die gleichzeitige Verwaltung mehrerer Zeichensätze oder/und mehrerer Ausgabe-Einheiten (z.B. sämtliche Grafik- und Text-Ausgaben auf dem Drucker oder in ein übertragbares Metafile etc.) oder die Grafik-Ausgabe in NDC-Koodinaten etc. möglich. Grundsätzlich ist das GDOS (Graphic-Device-Operating-System) dazu gedacht, GEM-Programme zwischen verschiedenen GEM-Rechnern (IBM, Apple, Atari) übertragbar zu machen.

Das Thema "Metafile" ist zu umfassend, um es hier näher beschreiben zu können. Ausführlichere Informationen finden Sie im Data Becker "Supergrafik"-Buch. Nun aber erst mal einige grundlegende Informationen zum Thema "GDOS".

Es gibt mehrere, immer wieder überarbeitete GDOS-Versionen. Die - nach meiner Kenntnis - neueste Version trägt die Bezeichnung "1.8". Die Versionen 1.1 ist in mancher Hinsicht mit Vorsicht zu genießen. Die Version 1.8 funktionierte bei meinen Tests fehlerfrei und ist bei der Drucker-Ausgabe erheblich schneller als 1.1.

Das A und O der GDOS-Anwendungen ist die ASSIGN.SYS-Datei. In dieser Header-Datei werden alle Informationen abgelegt, die zum Betrieb einer Workstation wichtig sind.

Hierin sind der Reihe nach die Treiber-Bezeichnungen und daran anschließend die Namen der jeweils dafür vorgesehenen Fonts eingetragen. Die Treiber-File-Namen enden mit der Extension .SYS und die Font-File-Namen mit .FNT. Die zweistellige Zahl vor dem Treibernamen stellt den Identifikator dar, der beim V_OPNWK()-Aufruf anzugeben ist:

01	Für Bildschirm.
11	Für Plotter.
21	Für Drucker.
31	Für Metafile.
41	Für Kamera.
51	Für Grafik-Tablett.

Der Id 01 ist dabei jedoch nur bei V_OPNVWK() einsetzbar, hat dann jedoch dieselbe Wirkung wie 04 (siehe unten).

Die vor den Kommentar-Zeilen stehenden Semikolons haben dieselbe Wirkung in ASSIGN.SYS wie das REM-Zeichen ' in GFA-Programmen: die damit beginnenden Zeilen werden ignoriert.

Ein ASSIGN.SYS-Datei ist eine "ganz normale" ASCII-Datei, die mit jedem besseren Textverarbeitungsprogramm problemlos geladen, bearbeitet und wieder abgespeichert werden kann und kann z.B. folgendermaßen aussehen:

```
;
path = D:\FONTS
;
; Der hier angegebene Suchpfad beschreibt dem GDOS den Weg zu
; den Treibern und - ggfs. - zu den unter den Treibernamen
; aufgeführten Fonts. Fehlt die Pfadangabe, so werden die Treiber-
; und Font-Files auf der aktuellen Hauptebene erwartet. Bei Angabe
; eines Pfads (hier: Ordner FONTS auf Station D) müssen die
; für den jeweiligen V_OPNWK()-Aufruf relevanten Treiber und Fonts
; unter diesem Pfad erreichbar sein.
;
; Die ersten vier Treiber-Bezeichnungen sind Dummy-Einträge, die
; nur den Zweck haben, die darunter angeführten Fonts der
; jeweiligen Bildschirm-Auflösung zuzuordnen.
;
; ##################################################
; Die Bedeutung des folgenden Treiber-Namens ist mir nicht
; bekannt. Bei meinen Tests hatte er grundsätzlich dieselbe
; Wirkung wie 04p screen.sys.
;
01p screen.sys
IBMHSS10.FNT
;
; ##################################################
; Die folgenden drei Treiber müssen im Treiber-File
; (s.o. path) nicht enthalten sein. Diese
; Treiber-Namen werden trotzdem vom GDOS in dieser Form
; erwartet. Der entsprechende Treiber ist im ROM enthalten.
; Die hier eingetragenen Fonts sind in GFA-BASIC verfügbar,
; sobald GDOS resident ist und - natürlich - die
; entsprechenden Fonts unter oben angegebenen Pfad
; auffindbar sind. In diesem Beispiel sind IBM-Fonts
; eingetragen, es können beliebige Font-Namen angegeben
; werden, solange die darunter abgelegten Fonts ein
; korrektes und brauchbares Format aufweisen.
; Der für die jeweilige Auflösung notwendige V_OPNVWK()-Aufruf
```

```
; wird von GFA-BASIC automatisch durchgeführt (siehe Beispiel
; unter VQT_EXTENT()).
;
;--------------------------
; Pseudo-Treiber für Lowres
02p screen.sys
IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT
;--------------------------
; Pseudo-Treiber für Midres
03p screen.sys
IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT
;--------------------------
; Pseudo-Treiber für Hires
04p screen.sys
IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT
;
; ##################################################
; 11 PLOTTER.SYS
; Falls ein Plotter-Treiber verfügbar ist, ist er
; hier einzutragen.
;
; ##################################################
; Nachfolgend wird die Treiber-Bezeichnung für den
; Drucker, sowie die Namen der verfügbaren Fonts
; erwartet. Hier ist z.B. der Treiber für
; den Epson-FX80 und Kompatible eingetragen.
; Es kann jedoch auch ebenso gut ein anderer
; Treiber angegeben sein (z.B. PRINTER.SYS
; für Atari-Laser oder FX85.SYS); das hängt jeweils
; davon ab, welchen Drucker Sie betreiben möchten,
; bzw. welcher Treiber verfügbar ist. Z.B. kann ein
; FX80-Treiber ohne weiteres auch für den FX85
; eingesetzt werden (nicht umgekehrt), obwohl dann
; auch ein FX85 nur mit einer Auflösung von ca.
; 960 mal 1420 Punkten läuft.
;
21 FX80.SYS
EPSHSS10.FNT
EPSHSS14.FNT
EPSHSS20.FNT
EPSHSS28.FNT
EPSHSS36.FNT
;
; ##################################################
;
; Hier wird der Treiber-File-Name des Metafile-Treibers
; eingetragen. Metafiles ermöglichen auch die Verarbeitung
; von Grafiken im NDC-Format (siehe VDIBASE), sind jedoch dann
; aufgrund ihres gewaltigen Speicherbedarfs nur mit ausgefeilten
; Tricks einsetzbar.
```

```
;
;
31 META.SYS
IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT
;
; ##################################################
; 41 KAMERA.SYS;      willkürlich gewählter Name
; Falls ein Kamera-Treiber verfügbar ist, ist er
; hier einzutragen.
;
; ##################################################
; 51 TABLETT.SYS;     willkürlich gewählter Name
; Falls ein Grafik-Tablett-Treiber verfügbar ist, ist
; er hier einzutragen.
```

Dem GDOS ist es übrigens egal, ob die Namen-Strings in Groß- und/oder Kleinbuchstaben geschrieben werden.

Version 3.0

GDOS? — GDOS resident?

Var=GDOS?

Liefert TRUE (-1), falls GDOS (ab Release 1.1) "resident" ist. Andernfalls wird FALSE (0) zurückgegeben.

Für diesen Befehl gibt es keinen V2.xx-Ersatz, da die Existenz von GDOS nur über einen internes Flag ermittelt werden kann (das mir nicht bekannt ist) oder über die Ausführung einer VDI-Workstation-Funktion. Im zweiten Fall könnte anhand der korrekten bzw. unkorrekten Ausführung festgestellt werden, ob GDOS vorhanden ist. Für diese zweite Möglichkeit lege ich allerdings meine Hand nicht ins Feuer, weshalb ich auch diesbezüglich kein Beispiel demonstriere. Wenn Sie mit dieser Möglichkeit arbeiten möchten, so bleibt das allein Ihr Risiko.

Version 3.0

V~H **Aktuelles VDI-Handle liefern**
V~H= {V~} **... setzen**

Var=V~H **(lesen)**
V~H=Handle **(schreiben)**

Liefert im ersten Fall das aktuell von GFA-BASIC verwendete VDI-Handle. In der Zuweisungsform ist es möglich, dieses Handle auch selbst zu bestimmen.

Handle:	Bedeutung:
>= 0	Freie Bestimmung des - bei VDISYS-Aufrufen anzugebenden - aktuellen Handles (siehe CONTRL+12 bzw. CONTRL(6)). Nach einem V_OPNWK() bzw. V_OPNVWK()-Aufruf ist das von diesen Funktionen zurückgegebene VDI-Handle durch V~h=V_OPN[V]WK()-Rückgabe zu initialisieren.
= -1	Das aktuelle VDI-Handle wird wieder auf den - beim internen GFA-V_OPNWK() vergebenen - Wert gesetzt. Anschließend erfolgen alle VDI-Ausgaben wieder ungepuffert auf dem Bildschirm. Dies ist auch der Fall, wenn eine andere Workstation noch offen ist. Es kann also durch V~h=Handle und V~h=-1 beliebig zwischen den Workstations und der virtuellen Standard-Ausgabe umgeschaltet werden.

Diese Funktion läßt sich in V2.xx einfach dadurch realisieren, indem nach BASIC-Start durch Gfa_handle=DPEEK(CONTRL+12) oder Gfa_handle=DPEEK(VDIBASE+40) das aktuelle GFA-Handle gelesen und zwischengespeichert wird und dann bei Bedarf statt V~H=-1 (in V3.0) der V2.xx-Poke DPOKE CONTRL+12,Gfa_handle ausgeführt wird.

Die Zuweisung der einzelnen Workstation-Handels erfolgt dann auf die gleiche Art und Weise: DPOKE CONTRL+12,Workstation_handle.

Version 3.0

V_CLRWK() Workstation-Puffer löschen

Var=V_CLRVWK()

Löscht den aktuellen Inhalt des Ausgabe-Puffers bzw. den Bildschirm der Workstation mit dem z. Zt. in V~H liegenden Handle. Auf dem Drucker wird zusätzlich ein FormFeed ausgeführt. Dieser Befehl hat auf die aktuelle Grafikseite grundsätzlich dieselbe Wirkung wie normalerweise der CLS-Befehl auf den Bildschirm.

Als V2.xx-Prozedur:

```
@v_clrwk
PROCEDURE v_clrwk
  VDISYS 3
RETURN
```

Version 3.0

V_CLSVWK() Virtual_Workstation schließen

Var=V_CLSVWK()

Schließt einen mit V_OPNVWK() geöffneten Bildschirm - mit dem z. Zt. in V~H liegenden Handle - und setzt das interne VDI-Handle wieder auf den GFA-Wert (siehe V~H=-1).

Dieser Aufruf ist in GFA-BASIC eigentlich nicht nötig, da bei GFA-Start das GFA-BASIC automatisch für sich selbst den Bildschirm als virtuelle Workstation anmeldet. Durch V~h=-1 wird immer wieder das an GFA-BASIC vergebene VDI-Handle im CONTRL-Array restauriert.

Vor V_CLSWK() für eine mit V_OPNWK() geöffneten Station **muß** vorher eine ggfs. durch V_OPNVWK() geöffnete virtuelle Workstation durch V_CLSVWK() geschlossen werden.

Als V2.xx-Prozedur:

```
@v_clsvwk
DPOKE CONTRL+12,Gfa_handle
PROCEDURE v_clsvwk
  VDISYS 101
RETURN
```

Nach diesem Prozedur-Aufruf müssen Sie allerdings selbst das GFA-Handle in CONTRL+12 wieder eintragen (siehe unter V~H).

Version 3.0

V_CLSWK() Workstation schließen

Var=V_CLSWK()

Schließt eine mit V_OPNWK() geöffnete Workstation - mit dem z. Zt. in V~H liegenden Handle -, führt alle noch enthaltenen Puffer-Einträge aus und setzt das interne VDI-Handle wieder auf den GFA-Wert (siehe V~H=-1).

Ein Einsatz-Beispiel finden Sie unter V_OPNWK().

Als V2.xx-Prozedur:

```
@v_clswk
DPOKE CONTRL+12,Gfa_handle
PROCEDURE v_clswk
  VDISYS 2
RETURN
```

Auch hier muß das GFA-Handle nach Aufruf von Ihnen selbst in CONTRL+12 restauriert werden (siehe unter V~H).

Version 3.0

V_OPNVWK() Virtual_Workstation öffnen/Parameter setzen

Var=V_OPNVWK(Treiber_Id [,Linientyp,Linienfarbe,...
...Markertyp,Markerfarbe,Textstil,...
...Textfarbe,Füllstil,Füllmuster,...
...Füllfarbe,Koordinatenflag])

Öffnet den Bildschirm als Ausgabe-Gerät mit der Identifikationsnummer Treiber_Id und richtet ihn als Workstation ein. In Var wird nach Abschluß das VDI-interne Geräte-Handle (Device-Handle) geliefert (siehe CONTRL+12 bzw. CONTRL(6)). Über dieses Handle ist der Bildschirm künftig ansprechbar. VDI-Ausgaben mit diesem Handle (siehe CONTRL+12 bzw. CONTRL(6)) erfolgen dann direkt auf dem Bildschirm (ohne Puffer - kein V_UPDWK() nötig). Der Bildschirm läßt sich durch V_OPNWK() nicht öffnen.

Dieser Aufruf ist in GFA-BASIC eigentlich nicht nötig, da bei GFA-Start das GFA-BASIC automatisch für sich selbst den Bildschirm als

virtuelle Workstation anmeldet. Durch V~h=-1 wird immer wieder das an GFA-BASIC vergebene VDI-Handle im CONTRL-Array restauriert. Bei einem Monochrom-Bildschirm werden durch die Funktion in WORK_OUT(0-56) folgende Werte geliefert (jeweils ein Element):

```
639,399,0,372,372,3,7,0,6,8,1,24,12,2,10,1,2,
3,4,5,6,7,8,9,10,3,0,3,3,3,0,3,0,3,2,0,1,1,0,
2,2,1,1,1,2,5,4,7,13,1,0,40,0,15,11,120,88
```

Für die Options-Parameter gelten die Ausführungen zu V_OPNWK() analog.

Als V2.xx-Prozedur:

Hier sind allerdings alle Parameter zu übergeben und zusätzlich noch der Pointer auf eine Integer-Rückgabevariable, die nach Abschluß das Handle der virtuellen Workstation enthält (hier: *Handle%). Die aktuellen Workstation-Einstellungen können aus INTOUT+0 bis INTOUT+88, sowie aus PTSOUT+0 bis PTSOUT+22 (jeweils Dpeeks im Wordabstand siehe WORK_OUT()) ausgelesen werden.

```
@v_opnvwk(treiber_id,Linientyp,Linienfarbe,...
          ...Markertyp,Markerfarbe,Textstil,...
           ...Textfarbe,Füllstil,Füllmuster,...
          ...Füllfarbe,Koordinatenflag,*Handle%)
PROCEDURE v_opnvwk(id%,lt%,lf%,mt%,mf%,ts%,tf%,fs%,fm%,ff%,kf%,bk%)
  LOCAL d.buf$
  DPOKE GCONTRL+4,5
  GEMSYS 77
  DPOKE CONTRL+12,DPEEK(GINTOUT)
  DPOKE CONTRL+4,6
  DPOKE CONTRL+6,11
  DPOKE CONTRL+8,45
  d.buff$=MKI$(id%)+MKI$(lt%)+MKI$(lf%)+MKI$(mt%)+MKI$(mf%)+MKI$(ts%)
  d.buff$=d.buff$+MKI$(tf%)+MKI$(fs%)+MKI$(fm%)+MKI$(ff%)+MKI$(kf%)
  BMOVE VARPTR(d.buff$),INTIN,22
  VDISYS 100
  *bk%=DPEEK(CONTROL+12)
RETURN
```

Version 3.0

V_OPNWK() Workstation öffnen/Parameter setzen

**Var=V_OPNWK(Treiber_Id [,Linientyp,Linienfarbe,...
...Markertyp,Markerfarbe,Textstil,...
...Textfarbe,Füllstil,Füllmuster,...
...Füllfarbe,Koordinatenflag])**

Der Gerätetreiber mit der Identifikationsnummer Treiber_Id (siehe Id in der ASSIGN.SYS-Beschreibung) geladen und eine Workstation dafür eingerichtet. In Var wird nach Abschluß das VDI-Handle (Device-Handle) für diese Station geliefert (siehe CONTRL+12 bzw. CONTRL(6)). Über dieses Handle ist die Station künftig ansprechbar. VDI-Ausgaben erfolgen dann in den durch CONTRL+12 bzw. CONTRL(6) bestimmten Gerätepuffer (siehe V~H). Die gerätetypischen Einstellungen können mit der WORK_OUT()-Funktion oder über INTOUT() bzw. PTSOUT() ermittelt werden.

Die in der Optionsklammer angegebenen 10 Parameter können zusätzlich eingesetzt werden, wenn die - intern - bei V_OPNWK() in INTIN(1) bis INTIN(10) zu übergebenden Grundeinstellungen vorbestimmt werden sollen. Wird diese Möglichkeit nicht genutzt, so bleiben die aktuellen Grafik-Grundeinstellungen erhalten.

Die Default-Einstellungen sind:

```
~V_OPNWK(id,1,1,1,1,1,1,1,1,1,2)
```

Wird V_OPNWK() verwendet, ohne daß das GDOS geladen wurde, führt dies zum System-Absturz. Das gleiche geschieht, wenn ein unzulässiger Id angegeben wird (z.B. 1 für Bildschirm). Der Bildschirm kann nur über V_OPNVWK() mit dem jeweils gültigen Auflösungs-Id (XBIOS(4)+2) geöffnet werden.

Fontname : Dutch
Fontname : Dutch
Fontname : Dutch
Fontname : Dutch
Fontname : Dutch
Fontname : Dutch
Fontname : Dutch

Fontname : Typewriter
Fontname : Typewriter
Fontname : Typewriter
Fontname : Typewriter
Fontname : Typewriter
Fontname : Typewriter
Fontname : Typewriter

Fontname : Swiss
Fontname : Swiss
Fontname : Swiss
Fontname : Swiss
Fontname : Swiss
Fontname : Swiss
Fontname : Swiss

Fontname : Fraktura
Fontname : Fraktura
Fontname : Fraktura
Fontname : Fraktura
Fontname : Fraktura
Fontname : Fraktura
Fontname : Fraktura

Beispiel:

```
IF OUT?(0)                   ! Drucker angeschlossen?
  IF GDOS?                   ! GDOS resident?
    RESERVE 10000            ! Speicher für Workstation freigeben
```

```
work_handle%=V_OPNWK(21)     ! Drucker-Workstation einrichten
x_faktor=INTOUT(0)/640       ! X-Koordinaten-Faktor in Hires
y_faktor=INTOUT(1)/400       ! Y-Koordinaten-Faktor in Hires
IF work_handle%              ! Station fehlerlos geöffnet?
  V~H=work_handle%           ! Neues Handle an VDI übergeben
  '
  ' ab jetzt gehen alle Grafik-Ausgaben auf den Workstation-
  ' Puffer:
  '         Breite der Grafikseite = INTOUT(0)
  '         Höhe der Grafikseite   = INTOUT(1)
  '
  ' Auf dem Bildschirm ist dies nicht zu erkennen.
  '
  CLIP 0,0,INTOUT(0),INTOUT(1)    ! Clip-Rechteck setzen
  font_anzahl%=VST_LOAD_FONTS(0) ! Alle Drucker- GDOS-Fonts,
  ' die in ASSIGN.SYS eingetragen und verfügbar sind,
  ' werden geladen.
  PRINT "GRAFIK wird bearbeitet ! (";
  PRINT font_anzahl%;") Fonts nachgeladen)"
  GRAPHMODE 2                ! Transparentmodus
  FOR i%=0 TO 10             ! ----------------------------.
    DEFFILL ,2,RANDOM(12)+8                                !
    PBOX RANDOM(60*i%)*x_faktor,RANDOM(36*i%)*y_faktor,... !-Grafik
    ...RANDOM(60*i%+40)*x_faktor,RANDOM(36*i%+40)*y_faktor ! Ausgabe
   NEXT i%                                                 !
   CIRCLE 320*x_faktor,200*y_faktor,200*y_faktor !----------'
   FOR i%=1 TO font_anzahl%+1                    !----------.
     ' Der Index der nachgeladenen Fonts beginnt mit 1, da  !
     ' auf dem 0-Index der aktuelle Standard-ROM-Font liegt.!
     font_index%=VQT_NAME(i%,font_name$) ! DEFTEXT-Index    !
     ' Face (siehe unter DEFTEXT) ermitteln (s. VQT_NAME()) !
     DEFTEXT ,0,,13,font_index%          ! Font variieren   !
     TEXT 10,i%*280+65,"Fontname : "+font_name$             !
     DEFTEXT ,1,,13,font_index%          ! Font variieren   !
     TEXT 10,i%*280+40+65,"Fontname : "+font_name$          !
     DEFTEXT ,4,,13,font_index%          ! Font variieren   !
     TEXT 10,i%*280+80+65,"Fontname : "+font_name$          !
     DEFTEXT ,5,,13,font_index%          ! Font variieren   ! Text-
     TEXT 10,i%*280+120+65,"Fontname : "+font_name$         !Ausgabe
     DEFTEXT ,17,,13,font_index%         ! Font variieren   !
     TEXT 10,i%*280+160+65,"Fontname : "+font_name$         !
     DEFTEXT ,20,,13,font_index%         ! Font variieren   !
     TEXT 10,i%*280+200+65,"Fontname : "+font_name$         !
     DEFTEXT ,16,,4,font_index%          ! Font variieren   !
     TEXT 10,i%*280+240+65,"Fontname : "+font_name$         !
   NEXT i%                   !------------------------------'
   ~V_UPDWK()                ! Grafikseite ausgeben
   ~VST_UNLOAD_FONTS(0)      ! Font-Speicher wieder freigeben.
   '                         ! Ab jetzt sind die Fonts nicht
   '                         ! mehr ansprechbar.
   ~V_CLSWK()                ! Workstation wieder schließen.
 ELSE                        ! V_OPNWK()-Fehler aufgetreten
   PRINT "Workstation konnte nicht geöffnet werden!"
   PRINT "Mögliche Fehler:"
   PRINT "keine ASSIGN.SYS gefunden"
   PRINT "kein Treiber gefunden"
   PRINT "zu wenig Speicher frei"
 ENDIF
 RESERVE                     ! BASIC-Speicher restaurieren
```

```
    ELSE                        ! GDOS ist nicht resident!
      PRINT "Sie haben vergessen, GDOS.PRG in den Auto-Ordner zu tun,"
      PRINT "bzw. Sie haben nicht die richtige Boot-Diskette eingelegt,"
      PRINT "bzw. Sie haben gar kein GDOS?"
    ENDIF
  ELSE                          ! Drucker ist nicht angeschlossen
    PRINT "Der Drucker ist nicht angeschlossen, bzw. nicht OnLine!"
  ENDIF
```

Sollen die GDOS-Fonts auf dem Bildschirm ausgegeben werden, ist weiter nichts zu tun, als durch VST_LOAD_FONTS(0) die Fonts zu laden, durch VQT_NAME() deren Index zu ermitteln und nach Abschluß durch VST_UNLOAD_FONTS(0) den Font-Speicher wieder freizugeben. V_UPDWK() und V_CLSWK() sind hier unnötig, da die Station generell geöffnet ist und die Ausgaben ohne Puffer direkt auf dem Bildschirm erfolgen (siehe Beispiel zu VQT_EXTENT()).

Wie oben bereits erwähnt, wird der für die jeweilige Auflösung notwendige V_OPNVWK()-Aufruf von GFA-BASIC automatisch durchgeführt. Zum Speicher ist zu sagen, daß eine Workstation immer mindestens soviel Speicher benötigt, daß für jeden Grafikpunkt ein Bit verfügbar ist und zusätzlich noch einen Arbeitspuffer von ca. 20 Prozent. In INTOUT(0) und INTOUT(1) werden nach der Stations-Eröffnung die Breite und Höhe der jeweiligen Grafikseite geliefert. Der Speicherbedarf errechnet sich dann aus (Höhe*Breite/8)+((Höhe*Breite/8)/100*20). Ein Atari-Laser-Drucker mit einer Auflösung von 2335 mal 3385 Punkten benötigt demnach

```
      2335*3385/8 =>    987.996
  +   987996/100*20 =>  197.599
                    -------------
                    =  1.185.595  Bytes
```

Drucker dieser Größenordnung lassen sich also nur mit einem Mega ST 2 bzw -4 oder mit einem auf mindestens 2 Megabyte aufgerüsteten 520ST+/1040STF betreiben. Selbst ein Epson-FX80 benötigt nach dieser Formel ca. 310.000 Bytes Speicherplatz für eine Grafikseite.

Im obigen Beispiel werden Grafikausgaben auf die Standard-Hires-Koordinaten (0 - 639/0 - 399) gesetzt. Die Anpassung an die aktuelle Workstation-Auflösung erfolgt durch die Multiplikation der Standard-Koordinaten mit dem vorher errechneten Auflösungs-Verhältnis (Realwerte: x_faktor/y_faktor). Die TEXT-Ausgabe habe ich davon ausgenommen, um zu zeigen, daß die Koordinaten der Grafikseite mit den normalen Bildschirm-Koordinaten nicht vergleichbar sind. Beim Testlauf hatte ich z.B. sechs Fonts zur Verfügung. Die letzte TEXT-

Ausgabe im 7. Schleifendurchlauf hatte also die Y-Koordinate 7*280*+240+65 = 2265.

Wie in den DEFTEXT-Zeilen gezeigt, kann auch der GDOS-Text in allen 32 VDI-Textarten und 26 VDI-Texthöhen beliebig variiert werden (siehe Grafik).

Übrigens ist bei Schließung der Workstation möglich, daß der Speicher nicht wieder korrekt restauriert werden kann, wenn seit der Öffnung keine Grafik-Ausgaben vorgenommen wurden. Eine Erklärung dafür habe ich zwar nicht, aber bei meinen Tests wurde das Programm immer mit einem "Fehlerhaftes RESERVE" quittiert, wenn die Station einfach nur geöffnet und anschließend ohne Ausgabe wieder geschlossen wurde. Der Speicher wurde vom GEM also nicht wieder freigegeben. Des weiteren kann es auch zu internen Fehlfunktionen führen (Maus inaktiv, Direktmodus nicht aufrufbar etc.), wenn nach Grafik-Ausgaben in den Puffer die Station ohne V_UPDWK()-Aufruf geschlossen werden soll. Die Reihenfolge der Befehle ist also nur in engen Grenzen variabel.

Als V2.xx-Prozedur:

```
' Der Aufruf ist identisch mit dem unter V_OPNVWK() beschriebenen
' Prozedur-Aufruf (siehe auch WORK_OUT()-Anmerkung oben).
PROCEDURE v_opnwk(id%,lt%,lf%,mt%,mf%,ts%,tf%,fs%,fm%,ff%,kf%,bk%)
  LOCAL d.buf$
  DPOKE GCONTRL+4,5
  GEMSYS 77
  DPOKE CONTRL+12,DPEEK(GINTOUT)
  DPOKE CONTRL+4,6
  DPOKE CONTRL+6,11
  DPOKE CONTRL+8,45
  d.buff$=MKI$(id%)+MKI$(lt%)+MKI$(lf%)+MKI$(mt%)+MKI$(mf%)+MKI$(ts%)
  d.buff$=d.buff$+MKI$(tf%)+MKI$(fs%)+MKI$(fm%)+MKI$(ff%)+MKI$(kf%)
  BMOVE VARPTR(d.buff$),INTIN,22
  VDISYS 1
  *bk%=DPEEK(CONTROL+12)
RETURN
```

Version 3.0

V_UPDWK() — Workstation-Puffer ausgeben

Var=V_UPDWK()

Veranlaßt die Ausgabe der aktuellen Workstation-Grafikseite auf dem Ausgabegerät (Workstation). Dabei gilt die Workstation als aktuell, deren Handle momentan in V~H liegt.

Bei Bildschirm-Workstations (V_OPNVWK()) ist diese Funktion überflüssig, da hier die Ausgaben direkt ausgeführt werden.

Als V2.xx-Prozedur:

```
@v_updwk
PROCEDURE v_updwk
  VDISYS 4
RETURN
```

Version 3.0

VQT_EXTENT() GEM-String-Ausmaße berechnen

Var=VQT_EXTENT(Text$ [,Xlo,Ylo,Xro,Yro,Xru,Yru,Xlu,Ylu])

Berechnet auf der Grundlage der Attribute des aktuellen GEM-Fonts (siehe DEFTEXT) die Ausmaße des in Text$ angegebenen Strings. Werden die acht optionalen Rückgabe-Variablen verwendet, werden in ihnen die nullpunkt-bezogenen Koordinaten des stringumfassenden Rechtecks geliefert:

Xlo/Ylo	Linke obere Ecke (immer 0/0).
Xro/Yro	Rechte obere Ecke.
Xru/Yru	Rechte untere Ecke.
Xlu/Ylu	Linke untere Ecke.

Die Koordinaten können in derselben Reihenfolge auch aus PTSOUT(0 - 7) ausgelesen werden. Der Text$ wird intern in seiner vollen Länge in INTIN(1) bis INTIN(n) (jeweils im LowByte) an die Funktion übergeben.

Da die Xlo/Xlo-Koordinate der Umfassungsbox-Box von dieser Funktion immer auf den Bildschirm-Nullpunkt gelegt werden, sind die Ordinaten Xru und Yru gleichbedeutend mit der Breite bzw. Höhe der Box.

Beispiel:

```
ON BREAK GOSUB break        ! Break abfangen
RESERVE 10000               ! Speicher für Fonts freigeben
a%=VST_LOAD_FONTS(0)        ! Alle in ASSIGN.SYS für die
'                           ! aktuelle Auflösung eingetragenen
'                           ! Fonts laden:
'                           !     Hires  = '02p screen.sys
'                           !     Midres = '03p screen.sys
'                           !     Lowres = '04p screen.sys
GRAPHMODE 3                 ! XOR-Modus
DO                          ! Endlos-Schleife (Exit nur durch Break)
```

```
    FOR j%=1 TO a%+1         ! Alle nachgeladenen Fonts
      font%=VQT_NAME(j%,a$) ! Font-Index und -Name ermitteln
      CLS                    ! Bildschirm klar
      DEFTEXT ,21,,26,font% ! Font einstellen
      ~VQT_EXTENT(a$)        ! Umfassungs-Box berechnen
      breite%=PTSOUT(4)      ! Xru = Breite (s.o)
      hoehe%=PTSOUT(5)       ! Yru = Höhe (s.o)
      PRINT AT(1,15);"Breite : ";breite%''"Höhe : ";hoehe%''''''
      x%=100                 ! Text-Ausgabe-X-Koordinate
      y%=100                 ! Text-Ausgabe-Y-Koordinate
      TEXT x%,y%,a$          ! Text ausgeben
      FOR i%=0 TO 60 STEP 1 ! -----------------------.
        BOX x%-i%,y%-hoehe%-i%,x%+breite%+i%,y%+i%     !
      NEXT i%                                          !- Umfassungs-
      FOR i%=60 TO 0 STEP -2                           !  Box-Effekt
        BOX x%-i%,y%-hoehe%-i%,x%+breite%+i%,y%+i%     !
      NEXT i%                ! -----------------------'
    NEXT j%                  ! Nächsten Font
  LOOP
  PROCEDURE break            ! Break-Abfang-Routine
    ' Diese Prozedur ist hier notwendig, um zu gewährleisten,
    ' daß der Font-Speicher korrekt wieder freigegeben, und
    ' der BASIC-Speicher wieder restauriert werden kann.
    ~VST_UNLOAD_FONTS(0)     ! Fonts abmelden
    ON BREAK                 ! In V3.0 bleibt ON BREAK GOSUB
    '                        ! auch nach Programmende aktiv.
    '                        ! Deshalb muß hier ON BREAK gesetzt werden.
    RESERVE                  ! BASIC-Speicher auf Maximum setzen.
    EDIT
  RETURN
```

Als V2.xx-Prozedur:

```
DEFTEXT ,16,,16
a$="V2.xx-VQT_EXTENT-Test"
@vqt_extent(a$,*x1%,*y1%,*x2%,*y2%,*x3%,*y3%,*x4%,*y4%)
TEXT 100,100,a$
BOX 100,100-y3%,100+x3%,100
'
PROCEDURE vqt_extent(txt$,vx1%,vy1%,vx2%,vy2%,vx3%,vy3%,vx4%,vy4%)
  LOCAL i%
  DPOKE CONTRL+4,8
  DPOKE CONTRL+6,LEN(txt$)       ! INTIN-Länge = Textlänge
  FOR i%=0 TO LEN(txt$)-1        ! String an INTIN übergeben
    DPOKE INTIN+i%*2,ASC(MID$(txt$,i%+1,1))
  NEXT i%
  VDISYS 116
  *vx1%=DPEEK(PTSOUT)
  *vy1%=DPEEK(PTSOUT+2)
  *vx2%=DPEEK(PTSOUT+4)
  *vy2%=DPEEK(PTSOUT+6)
  *vx3%=DPEEK(PTSOUT+8)
  *vy3%=DPEEK(PTSOUT+10)
  *vx4%=DPEEK(PTSOUT+12)
  *vy4%=DPEEK(PTSOUT+14)
RETURN
```

Version 3.0

VQT_NAME() GDOS-Font-Name/-Kennung liefern

Face=VQT_NAME(Index,Name$)

Liefert in Face (siehe DEFTEXT) die VDI-Kennung eines im zuletzt durch VST_LOAD_FONTS() geladenen Zeichensatz-Block enthaltenen Fonts. In Index wird dabei der Index des Fonts in der entsprechenden ASSIGN.SYS-Liste angegeben.

Name$ ist eine Rückabe-String-Variable, die nach Abschluß den im betreffenden Font-Header eingetragenen Font-Namen enthält. Ist GDOS nicht resident, wird der Wert 1 (Standard), sowie in Name$ der Name des aktuellen System-Zeichensatzes (6x6-, 8x8- bzw. 8x16 System-Font) geliefert.

Anwendungsbeispiele finden Sie unter V_OPNWK(), sowie unter VQT_EXTENT().

Als V2.xx-Prozedur:

```
@ vqt_name(Index,*Name$,*Face%)
DEFTEXT ,,,,Face%
' Diese Prozedur liefert dasselbe Ergebnis wie die
' V3.0-VQT_NAME()-Funktion. Allerdings muß hier
' statt des Rückgabe-String-Variable ein Pointer
' auf eine Rückgabe-String-Variable übergeben werden
' und zusätzlich noch ein Pointer auf eine Integer-
' Rückgabevariable, die nach Abschluß das benötigte
' GEM-Text-Face enthält.
' Installiert wird das Text-Face auch hier über
' DEFTEXT (siehe dort).
PROCEDURE vqt_name(ind%,f.n%,fc%)
  LOCAL nm$,i%,i.ntout%
  DPOKE CONTRL+6,1          ! INTIN-Länge
  DPOKE CONTRL+8,33         ! INTOUT-Länge
  DPOKE INTIN,ind%          ! ASSIGN.SYS-Index übergeben
  VDISYS 130                ! VDI-VQT_NAME-Aufruf
  i.ntout%=LPEEK(&H294A)    ! Internes Intout-Feld ermitteln
  '                         ! (siehe unter L~A).
  *fc%=DPEEK(i.ntout%)      ! Face aus internem Intout+0 lesen
  FOR i%=0 TO 31            ! 32 Zeichen Font-Name aus internem
    nm$=nm$+CHR$(PEEK(i.ntout%+3+i%*2)) ! Intout-Feld lesen
    '                       ! Das hier verwendete Intout ist
    '                       ! das Original-Intout-Feld des GEM,
    '                       ! nicht das des GFA-BASIC!
  NEXT i%
  *f.n%=nm$                 ! Font-Namen zurückgeben
RETURN
```

Version 3.0

VST_LOAD_FONTS() GDOS-Font laden

Var=VST_LOAD_FONTS(0)

Lädt die in ASSIGN.SYS angegebenen zusätzlich GEM-Zeichensätze in den nächstmöglichen freien Speicher (ggfs. vorher durch RESERVE freimachen - Malloc wird von der Funktion ausgeführt) und gibt in Var die Anzahl der tatsächlichen geladenen Fonts zurück. Wurde kein Font geladen, enthält Var anschließend den Wert Null. Der Null-Parameter ist von Digital Research vorgegeben und dient als Platzhalter für evtl. neue GEM-Versionen.

Anwendungsbeispiele finden Sie unter V_OPNWK(), sowie unter VQT_EXTENT().

Als V2.xx-Prozedur:

```
@vst_load_fonts(*fontanzahl%)
PROCEDURE vst_load_fonts(anz%)
  DPOKE CONTRL+6,1
  DPOKE INTIN,0
  VDISYS 119
  *anz%=DPEEK(INTOUT)
RETURN
```

Version 3.0

VST_UNLOAD_FONTS() GDOS-Font löschen

Var=VST_UNLOAD_FONTS(0)

Löscht die durch VST_LOAD_FONTS() geladenen Fonts, bzw. veranlaßt das GEM, ihre Existenz zu vergessen. Der dadurch belegte Speicher wird kann wieder vergeben werden (Mfree wird von der Funktion - falls möglich - automatisch ausgeführt).

Der Null-Parameter ist von Digital Research vorgegeben und dient als Platzhalter für evtl. neue GEM-Versionen.

Anwendungsbeispiele finden Sie unter V_OPNWK(), sowie unter VQT_EXTENT().

Als V2.xx-Prozedur:

```
@vst_unload_fonts(0)
PROCEDURE vst_unload_fonts(dummy%)
```

```
    DPOKE CONTRL+6,1
    DPOKE INTIN,0
    VDISYS 120
  RETURN
```

17.1 Verwendung eigener Fonts

Einige von Ihnen (vielleicht sogar sehr viele) haben evtl. kein gesteigertes Interesse daran, die eben erläuterten Befehle einzusetzen (keine Fonts, zu komplizierter Font-Header-Aufbau, zu komplizierte Aufruf-Bedingungen etc.).

Diesen Lesern möchte ich hier drei weitere Möglichkeiten vorstellen, innerhalb ihrer Programme eigene Fonts zu verwenden.

Als erstes finden Sie ein Maschinen-Programm, das vom Desktop aus oder aus dem Auto-Ordner gestartet werden kann.

Für die, die es noch nicht wissen:

> Ein Auto-Ordner ist ein Disketten-Ordner mit dem Namen AUTO. Darin enthaltene TOS-, TTP-, PRG-Programme werden automatisch gestartet, sobald dieser Ordner auf der bei Systemstart im Laufwerk liegenden Diskette enthalten ist. Als Grundbedingung gilt, daß diese Programm keine GEM-Aufrufe beinhalten dürfen. Ein- und Ausgaben sind also ausschließlich mit TOS-Befehlen (GEMDOS, BIOS, XBIOS) möglich. Dies hat seinen Grund darin, daß sich bei Ausführung dieser Auto-Programme das GEM noch nicht im RAM etabliert hat.
>
> Es gibt Trick-Programme, die diesen Mangel umgehen, indem Sie über den VBL-Vektor (siehe Kapitel 22 "System-Variablen") eine Warte-Routine solange als Interrupt ausführen lassen, bis das GEM installiert ist. Ist dies geschehen, führt diese VBL-Routine die in einem anderen (vorher bestimmten) Quasi-Auto-Ordner liegenden GEM-Programme der Reihe nach aus.

Einen solchen Trick setzt auch das folgende Programm ein. Allerdings nicht, weil es GEM-Befehle verwendet, sondern weil es auf das GEM warten muß, um die davon eingetragenen System-Font-Zeiger anschließend mit einer eigenen Font-Daten-Adresse überschreiben zu können.

Das Programm tut folgendes:

Es läßt sich vom System einen Bereich von 5000 Bytes zuweisen, lädt den auf der Hauptebene der aktuellen Diskette liegenden 8x16-System-Font mit Namen SYSTEM.FNT (inkl. Header) in diesen Speicherbereich und sichert ihn durch Keep Process (s. GEMDOS(49)), so daß er gegen weitere Speicherzugriffe geschützt ist.

Außerdem wird die Startadresse des geschützten Speichers in das letzte RAM-Longword geschrieben. So ist der Speicher auch verfügbar, wenn kein Font geladen werden soll. Der Speicher bleibt nämlich auch reserviert, wenn das SYSTEM.FNT-File nicht auf der Diskette gefunden wird. Das letzte RAM-Longword kann aus Adresse &H42E ausgelesen werden. Durch

```
Start%=Lpeek(Lpeek(&H42E)-4)
```

kann also die Startadresse des 5-KByte-Speichers ermittelt werden. Die eigentlichen Font-Grafikdaten liegen dann ab Start+602 (602 Byte für den Font-Header liegen vor den Font-Daten). Ein 8x16-System-Font-File erkennen Sie daran, daß es grundsätzlich 4698 Bytes lang ist und üblicherweise mit der Extension .FNT endet.

Nachdem das getan ist, installiert das Programm im VBL-Vektor eine Warte-Routine, die im Interrupt ständig zwei bestimmte Adressen kontrolliert:

&H607E	**RAM-Zeiger auf die vom GEM verwendeten Hires-Font-Daten.**
&H2924	**RAM-Zeiger auf die vom TOS verwendeten Hires-Font-Daten.**

Diese beiden Zeiger liegen im dazugehörigen Font-Header. Die System-Font-Header beginnen also 76 Bytes vor diesen Adressen.

Wurden diese Adressen nach Installation des GEM von diesem mit der Original-Font-Daten-ROM-Adresse (&HFD39B8) belegt, wird die Font-Daten-Adresse des nachgeladenen Fonts in diese beiden Adressen eingetragen, wodurch GEM und TOS nun mit diesem Font arbeiten. Anschließend löscht sich die VBL-Routine selbstständig aus dem VBL-Queue.

Das Programm ist ausführlich kommentiert, so daß es für all jene, die sich ein wenig in Maschinensprache auskennen, keine Verständnis-Schwierigkeiten geben dürfte.

Alle "Nicht-Assembler-Interressierten" finden im Anschluß an den Assembler-Source-Code einen sogenannten BASIC-Loader, der das Programm SYSTEM.PRG auf der Diskette erzeugt. Dieses Programm kopieren Sie dann also in den Auto-Ordner, legen ein 8x16-System-Font-File mit Namen SYSTEM.FNT auf die Hauptebene derselben Diskette und können nun den Reset-Knopf drücken. Falls der geladene Font nicht der Standard-Font war, können Sie nun den neuen Font auf dem Desktop bewundern. Werden die beiden Adressen später nicht mehr manipuliert, so bleibt der Font bis zum nächsten Reset erhalten.

Das Programm muß - wie gesagt - nicht unbedingt im Auto-Ordner gestartet werden, es kann ebensogut auch vom Desktop aus gestartet werden. Außerdem wird für den nachgeladenen Font der aktuell im RAM liegende 8x16-Header verwendet. Der mitgeladene Header des neuen Fonts bleibt unberücksichtigt. Soll auch der Header verändert werden, haben Assembler- Programmierer die Möglichkeit, den Source-Code dahingehend zu ändern, daß der neue Header (ab Start) mit einer move.w-Schleife in die Original-Header-Bereiche (&H607E-76 bzw. &H2924-76) kopiert wird. BASIC-Programmierer können aus dem BASIC heraus die ab Start liegenden 602 Bytes in die Original-Header-Bereiche kopieren:

```
Bmove Start%,&H607E-76,602
Bmove Start%,&H2924-76,602
```

Um den Original-Header evtl. restaurieren zu können, sollten Sie ihn zwischenspeichern und ggfs. wieder zurückkopieren.

```
;-------------------------------------------------------------
; AUTO-FONTLOADER (SEKA-Source-Code)                         |
; von Marcus Bode, 3200 Hildesheim                           |
;-------------------------------------------------------------
begin:                  ; ** Programm initialisieren **      |
  move.l    a7,a5       ; Stack-Pointer in A5 puffern        |
  move.l   4(a5),a5     ; Basepage-Anfang holen              |
  move.l   $c(a5),d0    ; Programm-Länge holen               |
  add.l    $14(a5),d0   ;DATA-Bereich addieren               |
  add.l    $1c(a5),d0   ;BSS-Bereich addieren                |
  add.l    #$100,d0     ; $100 Bytes für Basepage addieren   |
  move.l   d0,laenge    ; Gesamtlänge puffern                |
;-------------------------------------------------------------
super:                  ; ** Supervisor Mode einschalten **  |
  clr.l    -(sp)        ; User Stack wird Supervisor Stack   |
  move.w   #$20,-(sp)   ;SUPER-Opcode übergeben              |
  trap     #1           ; GEMDOS aufrufen                    |
  add.l    #6,sp        ; Stack aufräumen                    |
  move.l   d0,oldssp    ; alten Supervisor-Stack puffern     |
  move.l   $42e,a0      ; RAM-Endadresse holen, ...          |
  sub.l    #4,a0        ; ...Minus-Offset (4 Byte) und...    |
```

```
  move.l   #buf,(a0) ; ...Puffer-Adresse dorthin schreiben
;--------------------------------------------------------------
open:                ; ** Font-Datei öffnen **
  move.w   #2,-(sp)  ; Dateiattribut übergeben
  pea      filename  ; File-Namen-Adresse übergeben
  move.w   #$3d,-(sp) ;OPEN-Opcode übergeben
  trap     #1        ; GEMDOS aufrufen
  addq.l   #8,sp     ; Stack aufräumen
  tst.l    d0        ; Fehler aufgetreten?
  bmi      break     ; Ja, dann alles abbrechen!
;--------------------------------------------------------------
read:                ; ** Font-Datei einlesen **
  move.w   d0,handle ; File-Handle zwischenspeichern
  pea      buf       ; Font-Puffer-Adresse übergeben
  move.l   #4698,-(sp) ; System-Font-Länge übergeben
  move.w   handle,-(sp) ;File-Handle übergeben
  move.w   #$3f,-(sp) ; READ-Opcode übergeben
  trap     #1        ; GEMDOS aufrufen
  add.l    #12,sp    ; Stack aufräumen
  tst.l    d0        ; Fehler aufgetreten?
  bmi      break     ; Ja, dann alles abbrechen!
;--------------------------------------------------------------
close:               ; ** Font-Datei wieder schließen **
  move.w   handle,-(sp) ; File-Handle übergeben
  move.w   #$3e,-(sp) ; CLOSE-Opcode übergeben
  trap     #1        ; GEMDOS aufrufen
  addq.l   #4,sp     ; Stack aufräumen
  tst.l    d0        ; Fehler aufgetreten?
  bmi      break     ; Ja, dann alles abbrechen!
;--------------------------------------------------------------
change:              ; ** Vektoren ändern **
  move.l   #buf+$25a,$607e ; neue Font-Adresse in Header
  move.w   #32,d0    ; 8 mögliche VBL-Routinen mal 4 Bytes
  move.l   #$4ce,a0  ; Startadresse der VBL-Routinen holen
  move.w   #4,d1     ; erste VBL-Routine überspringen
;--------------------------------------------------------------
loop:                ; ** Freien VBL-Vektor suchen **
  tst.l    (a0,d1)   ; testen, ob Routine benutzt wird
  beq      free      ; Nein, dann Suche abbrechen
  addq     #4,d1     ; sonst nächste VBL-Routine
  cmp      d0,d1     ; schon alle Routinen getestet?
  bne      loop      ; Nein, dann weitersuchen
;--------------------------------------------------------------
error:               ; ** Error-Exit der Suchroutine **
  move.l   #0,laenge ; Gesamtlänge auf Null setzen
  bra      fertig    ; Alle Routinen getestet!
;--------------------------------------------------------------
break:               ; ** Error-Exit der Disk-Routinen **
  move.l   #0,d0     ; Gesamtlänge auf Null setzen
  bra      fertig    ; Programm beenden
;--------------------------------------------------------------
free:                ; ** Freie Routine gefunden **
  lea      (a0,d1),a2 ;Adresse der freien VBL-Routine holen
  move.l   a2,vbl    ; Adresse zwischenspeichern
  move.l   #instal,(a2) ; VBL-Vektor auf User-Routine setzen
;--------------------------------------------------------------
fertig:              ; ** Zurück in User Mode **
  move.l   oldssp,-(sp) ; Alten Supervisor Stack holen
  move.w   #$20,-(sp)   ; SUPER-Opcode übergeben
```

```
  trap      #1          ; GEMDOS aufrufen
  add.l     #6,sp       ; Stack aufräumen
  move.l    laenge,d0 ; Gesamtlänge zurückholen
;-------------------------------------------------------------
ende:                   ; ** Programm beenden **
  move.w    #0,-(sp)  ; Fehler-Code 0 übergeben
  move.l    d0,-(sp)  ; Gesamtlänge übergeben
  move.w    #$31,-(sp) ;KEEP PROCESS-Opcode übergeben
  trap      #1          ; GEMDOS aufrufen (zurück zum Desktop)
;-------------------------------------------------------------
instal:                 ; ** VBL-Interrupt-Routine **
  cmpi.l    #buf+$25a,$607e ; Font-Daten-Zeiger verändert?
  beq       weiter      ; Nein, dann weiter prüfen
  move.l    #buf+$25a,$607e;sonst neue Font-Datenadresse in
  move.l    #buf+$25a,$2924;TOS-/GEM- Font-Vektoren eintragen
  move.l    vbl,a0      ; Adresse der User-VBL-Routine holen
  clr.l     (a0)        ; User-VBL-Routine löschen
;-------------------------------------------------------------
weiter:                 ; ** VBL-Routine beenden **
  rts                   ; Rücksprung
;-------------------------------------------------------------
filename:   dc.b 'system*.fnt',0 ; File-Name des neuen Fonts
handle:     blk.w 1   ; Speicher für File-Handle
laenge:     blk.l 1   ; Speicher für Gesamtlänge
vbl:        blk.l 1   ; Speicher für VBL-Adresse
oldssp:     blk.l 1   ; Speicher für Supervisor Stack
buf:        blk.b 5000 ;Speicher für neuen System-Font
;-------------------------------------------------------------
```

```
'*************************************************
' BASIC-Loader erzeugt SYSTEM.PRG
'*************************************************
DO
  READ a$
  IF a$<>"*" AND a$<>"**"
    rout$=rout$+MKI$(VAL("&H"+a$))
  ELSE
    IF a$="*"
      rout$=rout$+STRING$(2500,MKI$(&HFFFF))
    ENDIF
  ENDIF
  EXIT IF a$="**"
LOOP
OPEN "O",#1,"SYSTEM.PRG"
BPUT #1,VARPTR(rout$),LEN(rout$)
CLOSE
'
DATA 601A,0,14DC,0,0,0,0,0,0,0,0,0,0,0
DATA 2A4F,2A6D,4,202D,C,D0AD,14,D0AD,1C,680
DATA 0,100,23C0,0,148,42A7,3F3C,20,4E41,DFFC,0,6
DATA 23C0,0,150,2079,0,42E,91FC,0,4,20BC,0,154
DATA 3F3C,2,4879,0,13A,3F3C,3D,4E41,508F,4A80,6B00,76
DATA 33C0,0,146,4879,0,154,2F3C,0,125A,3F39,0,146
DATA 3F3C,3F,4E41,DFFC,0,C,4A80,6B00,4C,3F39,0,146
DATA 3F3C,3E,4E41,588F,4A80,6B00,38,23FC,0,3AE,0,607E
DATA 303C,20,207C,0,4CE,323C,4,4AB0,1800,6700,22,5841
DATA B240,6600,FFF2,23FC,0,0,0,148,6000,1C,203C,0
DATA 0,6000,12,45F0,1800,23CA,0,14C,24BC,0,10E,2F39
DATA 0,150,3F3C,20,4E41,DFFC,0,6,2039,0,148,3F3C
DATA 0,2F00,3F3C,31,4E41,CB9,0,3AE,0,607E,6700,1E
```

```
DATA 23FC,0,3AE,0,607E,23FC,0,3AE,0,2924,2079,0
DATA 14C,4290,4E75,7379,7374,656D,2A2E,666E,7400,*
DATA FFFF,FFFF,FFFF,FFFF,FFFF,FFFF,FFFF,0,1A,1412,A14
DATA 60C,1814,2C18,606,1212,E0A,A00,**
```

Beachten Sie bitte, daß dieses Programm nur für das ROM-TOS vom 6.2.1986 gilt.

Die zweite Möglichkeit, eigene Fonts im Programm zu verwenden, besteht darin, nur für die Dauer des BASIC-Programms die oben erwähnten beiden Font-Zeiger zu "verbiegen". Die folgende Prozedur erledigt dies für Sie und liefert Ihnen nach erledigter Arbeit die Startadresse der Font-Daten. Der dazugehörige Font-Header beginnt 602 Bytes vor dieser gelieferten Adresse. Auch hier ist zu beachten, daß dieses Programm nur für das ROM-TOS vom 6.2.1986 gilt.

Das Besondere an dieser Routine ist, daß Sie auch Degas- und STAD-Fonts (STAD-Fonts sind identisch mit OLIFONT-Fonts) verarbeiten kann (jeweils Hires-8x16-Fonts!). Soll ein Degas-Font geladen werden, hat man die Möglichkeit, einen zweiten Font-File-Namen anzugeben. Diese zweite Font-Datei wird dann in den ASCII-Bereich von 128 - 255 hinzugeladen. Wird kein zweiter Font-Name angegeben, bleibt der ASCII-Bereich von 128 bis 255 unbelegt.

Ein Degas-Hires-Font besteht immer aus nur 128 Zeichen.

Der neue Font wird entweder direkt bei HIMEM (falls der RAM-Font-Pointer auf die Original-Font-Adresse im ROM zeigt) oder an der aktuellen Font-Adresse (falls schon durch SYSTEM.PRG ein neuer Font außerhalb des BASIC installiert wurde) abgelegt.

Im Falle, daß durch SYSTEM.PRG kein Font-Puffer angelegt wurde, wird der BASIC-Speicher jeweils am oberen Ende von jedem geladenen Font um 5000 Bytes reduziert.

Der BASIC-Speicher darf in diesem Fall - solange der geladene Font aktiv ist - nicht durch RESERVE vergrößert werden, da sonst die BASIC-Daten mit den Font-Daten kollidieren können. Das hätte zwar keinen Absturz zur Folge, es wären jedoch anschließend keine Texte mehr entzifferbar.

Im Beispiel wird der SYSTEM.FNT geladen. Degas- oder STAD-Aufrufe kann ich Ihnen hier leider nicht zeigen, da ich nicht weiß, über welche relevanten Fonts Sie verfügen. Ich hätte Ihnen gern einige

selbstentworfene Fonts auf der beiliegenden Diskette mitgeliefert, aber der Speicherplatz reichte leider nur noch für einen System-Font.

```
DEFTEXT ,0,,13
TEXT 100,50,"Alter Font"
@fontset(3,"SYSTEM.FNT","",*nfont%)
TEXT 100,100,"Neuer Font"
IF nfont%>BASEPAGE
  @original                        ! s.o.
  TEXT 100,150,"Wieder alter Font"
ENDIF
PROCEDURE fontset(typ%,font1$,font2$,font%)
  ' Typ%      Gibt den zu ladenden Font-Typ an:
  '              1  =  Degas
  '              2  =  STAD (bzw. OLIFONT)
  '              3  =  SYSTEM
  ' Font1$    Font-File-Name (bei Degas ggfs. für ASCII 0 bis 127)
  ' Font2$    Font-File-Name für 2. Degas-Halb-Font (ASCII 128-255)
  ' Font%     Pointer auf eine Integer-Rückgabevariable, die nach
  '           Abschluß die Startadresse der neuen Font-Daten enthält.
  LOCAL a$,a%,fnt%,i%,j%,k%
  IF typ%=1 OR typ%=2 OR typ%=3    ! Zulässige Typ-Angabe?
    IF EXIST(font1$)               ! Font-Datei existiert?
      a$=MKI$(&HA000)+MKL$(&H20094E75) ! Maschinen-Programm
      '                            ! (s. Prozedur Sysfont unter ASC)
      a%=VARPTR(a$)                ! Routinen-Adresse
      fnt%=LPEEK(&H607E)           ! Aktuelle Font-Adresse
      orig%=LPEEK(LPEEK(C:a%()+8)+76) ! Original-8x16-ROM-Font-Adresse
      IF fnt%=orig%                ! Aktueller Font = Original-ROM-Font?
        RESERVE FRE(0)-5000        ! 5-KB oberhalb von HIMEM freigeben
        fnt%=HIMEM+602             ! Header überspringen
        IF typ%<3                  ! Degas- oder STAD-Font?
          BMOVE orig%-602,HIMEM,602 ! Header in Font-Buffer kopieren
        ENDIF
      ENDIF
      IF typ%<3                    ! Degas-Font bzw. STAD-Font?
        FOR k%=0 TO 1              ! Evtl. 2 Durchläufe (bei Degas)
          b$=STRING$(2048*typ%,0)  ! Arbeitspuffer einrichten
          OPEN "I",#1,font1$       ! Datei öffnen
          BGET #1,VARPTR(b$),2048*typ% ! 2048 bzw. 4096 Bytes in
          '                        ! den Arbeits-Puffer laden
          CLOSE
          FOR j%=0 TO (128*typ%)-1 ! 128 (Degas)
                                   !  o. 256 (STAD) Zeichen
            FOR i%=0 TO 15         ! Je 16 Zeilen (Scan-Lines)
              POKE k%*128+fnt%+j%+i%*256,PEEK(VARPTR(b$)+i%+j%*16)
              ' .--------------------------.
              ' | Siehe unter "Font-Aufbau" |
              ' '--------------------------'
            NEXT i%
          NEXT j%
          IF k%=0                  ! Erster Durchlauf?
            IF typ%=1              ! Degas-Font?
              IF EXIST(font2$)=0   ! 2. Font-Datei existiert nicht?
                k%=1               ! Dann Schleife abbrechen
              ENDIF
            ELSE                   ! STAD-Font !
              k%=1                 ! Schleife abbrechen
```

```
          ENDIF
        ENDIF
      NEXT k%
    ENDIF
    IF typ%=3                          ! System-Font laden?
      BLOAD font1$,fnt%-602            ! Datei in Font-Puffer laden
    ENDIF
    *font%=fnt%                        ! Font-Daten-Adresse zurückgeben
    LPOKE &H2924,fnt%                  ! TOS-Font-Pointer verbiegen
    LPOKE &H607E,fnt%                  ! GEM-Font-Pointer verbiegen
  ENDIF
 ENDIF
RETURN
```

Wurden auf diese Art neue Fonts hinter HIMEM installiert, gehen Sie nach Verlassen des BASICs oder bei Voll-RESERVE verloren. Der Original-Font muß also vorher restauriert werden, da sonst auch hier später (auf dem Desktop) kein Text mehr erkennbar ist:

```
PROCEDURE original
  LOCAL a$,a%,orig%
  a$=MKI$(&HA000)+MKL$(&H20094E75) ! Maschinen-Programm
  a%=VARPTR(a$)                    ! Routinen-Adresse
  orig%=LPEEK(LPEEK(C:a%()+8)+76)  ! Original-Font-Adresse im ROM
  LPOKE &H2924,orig%               ! TOS-Font-Pointer restaurieren
  LPOKE &H607E,orig%               ! GEM-Font-Pointer restaurieren
  RESERVE XBIOS(2)-HIMEM-16384+FRE(0) ! BASIC-Speicher restaurieren
RETURN
```

War vor BASIC-Start schon ein neuer Font durch SYSTEM.PRG resident installiert, so kann das BASIC ohne weiteres wieder verlassen werden.

Font-Aufbau:

Die Degas- und STAD-Fonts sind so organisiert, daß die 16 Scan-Lines eines Zeichens - beginnend mit ASCII 0 - direkt nacheinander ab dem ersten Byte der Font-Datei abgelegt sind. Diesem Block wird bei Degas dann ein Word angehängt, das den Auflösungsindex enthält.

In den folgenden Schemen steht s für Scan-Line und z für ASCII-Zeichen.

```
-----------------------------------------------------------------
Degas-Font:
  s1z0-s2z0-s3z0-s4z0- .... -s14z127-s15z127-s16z127
  '------------------------.------------------------'
                           |
                           |
          2048  Bytes für die Font-Daten (16*128)
        +    2  Bytes für Auflösung (1=Hires/2=Midres/3=Lowres)
        --------
        = 2050  Bytes Font-File-Länge
```

```
 Ein Degas-Font in hoher Auflösung umfaßt immer nur 128 Zeichen!
----------------------------------------------------------------
STAD (bzw. OLIFONT-) Font:
   s1z0-s2z0-s3z0-s4z0- .... -s14z255-s15z255-s16z255
  '-----------------------.-----------------------'
                          !
                          !
            = 4096  Bytes für Font-Daten (16*256)
 Ein STAD-Font in hoher Auflösung umfaßt immer 256 Zeichen!
----------------------------------------------------------------
SYSTEM-Font:
   s1z0-s1z1-s1z2-s1z3- .... -s16z253-s16z254-s16z255
  '-----------------------.-----------------------'
                          !
                          !
              4096  Bytes für Font-Daten (16*256)
          +    602  Bytes für Font-Header
          --------
          =   4698  Bytes Font-File-Länge
 In einem Sytem-Font liegen dagegen immer die jeweiligen
 Scan-Lines aller 256 Zeichen nacheinander im Speicher:
----------------------------------------------------------------
```

In der oben eingesetzten Konvertierungs-Schleife werden Degas- und STAD-Fonts nach diesem Schema in einen 4096 Byte großen System-Font konvertiert, wobei beim Degas-Font im ersten Schleifendurchlauf (K%=0) die ASCIIs 0 bis 127 und evtl. im zweiten Durchlauf (K%=1) die ASCIIs 128 bis 255 belegt werden. Da beide Fonts über keinen Header verfügen, wird der aktuelle 8x16-System-Header vor die Font-Daten gehängt.

Die ganze Sache wäre nur die Hälfte wert, wenn man die so installierten neuen System-Fonts nicht auch als System-Font-File abspeichern könnte. So haben Sie die Möglichkeit, Ihre eigenen Fonts auf dem Degas- oder OLIFONT-Font-Editor zu kreieren, diese (bzw. bereits vorhandene Fonts) dann mit Setfont zu laden und anschließend mit der folgenden Prozedur als System-Font abzuspeichern.

```
PROCEDURE make_sys(font%,fname$)
  ' Font%  =  Die von Setfont gelieferte Font-Daten-Adresse
  ' Fname$ =  Name der neuen System-Font-Datei
  IF fname$>"" AND fname$<>"\" AND fname$<>"\.FNT"
    BSAVE fname$,font%-602,4698
  ENDIF
RETURN
```

Um das Bild abzurunden, folgen noch zwei Prozeduren, die den aktuell durch Setfont installierten Font aus dem System-Format in das Degas- bzw. STAD-Format konvertieren, wobei bei der Degas-Konvertierung nur die unteren 128 Zeichen (ASCII 0 bis 127) berücksichtigt werden.

```
PROCEDURE make_deg(font%,fname$)
  ' Font% =  Die von Setfont gelieferte Font-Daten-Adresse
  ' Fname$ =  Name der neuen Degas-Font-Datei
  LOCAL b$,i%,j%
  IF fname$>"" AND fname$<>"\" ! Gültiger Dateiname?
    b$=STRING$(2048,0)         ! Arbeitspuffer einrichten
    FOR j%=0 TO 127            ! 128 Zeichen
      FOR i%=0 TO 15           ! Je 16 Zeilen (Scan-Lines)
        POKE VARPTR(b$)+i%+j%*16,PEEK(font%+j%+i%*256)
        '                      ! Konvertieren
      NEXT i%
    NEXT j%
    b$=b$+MKI$(1)              ! Auflösungsindex anhängen
    BSAVE fname$,VARPTR(b$),2050 ! Degas-Font-Datei speichern
  ENDIF
RETURN
'
PROCEDURE make_stad(font%,fname$)
  ' Font% =  Die von Setfont gelieferte Font-Daten-Adresse
  ' Fname$ =  Name der neuen STAD-Font-Datei
  IF fname$>"" AND fname$<>"\" ! Gültiger Dateiname?
    b$=STRING$(4096,0)         ! Arbeitspuffer einrichten
    FOR j%=0 TO 255            ! 256 Zeichen
      FOR i%=0 TO 15           ! Je 16 Zeilen (Scan-Lines)
        POKE VARPTR(b$)+i%+j%*16,PEEK(font%+j%+i%*256)
        '                      ! Konvertieren
      NEXT i%
    NEXT j%
    BSAVE fname$,VARPTR(b$),4096 ! STAD-Font-Datei speichern
  ENDIF
RETURN

----------------------------------------------------------------
PATCH für Installation eigener Fonts innerhalb eines
V2.xx-GFA-Editors (nur gültig für ROM-TOS vom 6.2.86!)
----------------------------------------------------------------
```

Der GFA-BASIC-Interpreter hat die Eigenschaft, nach jeder Rückkehr zum Editor (vom Programm oder Direktmodus), den Original-System-Font zu restaurieren. Dadurch ist die Verwendung eigener Fonts innerhalb des V2.xx-Editors erschwert. Dieser Patch ändert die dafür zuständige Programmsequenz so, daß sich die Restauration immer auf den Font bezieht, dessen Adresse im Font-Pointer des Systems liegt. Dadurch sind auch innerhalb des V2.xx-Editors eigene Fonts einsetzbar.

Der V3.0-Editor benutzt dagegen eine andere - mir bisher schleierhafte - Methode, seinen Font bei Rückkehr aus dem Programm oder vom Direktmodus zu restaurieren. Wahrscheinlich geschieht dies über einen internen Vektor, dessen Lage und Bedeutung sich mir noch nicht offenbart hat.

```
a$=MKL$(&H43F90000)+MKI$(&H6032) ! Eigene Font-Adresse einschleusen
'
' Ass.:     Lea $6032,A1         ! TOS-Font-Header-Start nach A1
'
b$=MKL$(&HA0002269)+MKI$(&H8)    ! Original-System-Font-Adresse holen
'
' Ass.:     A000                 ! Line-A-Vektor holen
'           Move.L $0008(A1),A1  ! TOS-Font-Header-Start nach A1
'
al$="Aktiven Interpreter |oder|Original-'GFABASIC.PRG'|patchen?"
ALERT 1,al$,1,"SPEICHER|DISK-PRG",back%
IF back%=1                       ! Aktiven Speicher-Editor patchen
  ALERT 2,"Handelt es sich um:      ",1," V2.0 | V2.02 ",back2%
  IF back2%=1                    ! V2.0-Version
    BMOVE VARPTR(a$),BASEPAGE+256+48162,6 ! Neue Sequenz übertragen
    BMOVE VARPTR(a$),BASEPAGE+256+48258,6 ! Neue Sequenz übertragen
  ELSE                           ! V2.02-Version
    BMOVE VARPTR(a$),BASEPAGE+256+49358,6 ! Neue Sequenz übertragen
    BMOVE VARPTR(a$),BASEPAGE+256+49454,6 ! Neue Sequenz übertragen
  ENDIF
ELSE                             ! Disk-Programm patchen
  ' Vorsichtshalber vorher eine Sicherheitskopie anlegen!
  IF EXIST("GFABASIC.PRG")       ! GFABASIC.PRG gefunden?
    RESERVE XBIOS(2)-16384-HIMEM+FRE(0)-60000 ! 60 KByte freigeben
    BLOAD "GFABASIC.PRG",HIMEM   ! GFABASIC.PRG laden
    OPEN "U",#1,"GFABASIC.PRG"   ! GFABASIC-Programm-File öffnen
    FOR i%=0 TO LOF(#1) STEP 2   ! Gesamte Länge durchgehen
      POKE XBIOS(2)+(i%/2),255   ! Zeitvertreib
      IF MKI$(DPEEK(HIMEM+i%))+MKL$(LPEEK(HIMEM+i%+2))=b$
        '                        ! Alte Sequenz gefunden?
        SEEK #1,i%               ! File-Pointer setzen
        BPUT #1,VARPTR(a$),6     ! Neue Sequenz eintragen
      ENDIF
    NEXT i%
    CLOSE
    RESERVE XBIOS(2)-16384-HIMEM+FRE(0) ! Speicher restaurieren
  ELSE
    ALERT 3,"GFABASIC.PRG nicht gefunden!",1,"RETURN",back%
  ENDIF
ENDIF
```

GFA-BASIC-Spezial-Font

Nun folgt die wohl interessanteste Möglichkeit für die Verwendung eigener Fonts. Mit diesem Spezial-Font ist es möglich, mit (fast) beliebig vielen Fonts gleichzeitig zu arbeiten. Die Größe der Font-Zeichen ist nur (!) durch den verfügbaren Speicherplatz beschränkt. Die Prozedur Ptext erwartet einen Font in einem ganz bestimmten Format.

Der Font ist folgendermaßen aufzubauen:

```
Byte 1 und 2 sind 2 Header-Bytes:
    Byte 1 = Breite eines Zeichens in Pixel
    Byte 2 = Höhe eines Zeichens in Pixel
Daran anschließend liegen die einzelnen
Zeilen der Zeichen der Reihe nach im Speicher:
    Scan-Zeile 1 des 1. Zeichens (ASCII 0)
    Scan-Zeile 2 des 1. Zeichens (ASCII 0)
            .
    Scan-Zeile n des 1. Zeichens (ASCII 1)
    Scan-Zeile 1 des 2. Zeichens (ASCII 1)
    Scan-Zeile 2 des 2. Zeichens (ASCII 1)
            .
    Scan-Zeile n des 2. Zeichens (ASCII 1)
    .
    usw. bis
    .
    Scan-Zeile 1 des 256. Zeichens (ASCII 255)
    Scan-Zeile 2 des 256. Zeichens (ASCII 255)
    Scan-Zeile n des 256. Zeichens (ASCII 255)
```

Es handelt sich dabei um 256 GFA-PUT-Strings, die ohne die 6 Header-Bytes (da alle Zeichen gleiche Definitionen besitzen) nacheinander im Speicher liegen. Dabei ist wichtig, daß die Breite jeder der 256 Einzel-PUT-Images eine durch 2 teilbare Byte-Breite hat (siehe 9.5.1 "Organisation eines PUT-Strings"). Die Größe des Zeichens innerhalb des Einzelblocks ist dagegen unerheblich.

Bezüglich der Bildschirmauflösung ist darauf zu achten, daß jeweils entsprechende Fonts verwendet werden (Hires = 1 Farbebene/Midres = 2 Farbebenen/Lowres = 4 Farbebenen).

```
DIM fontbuff%(25000)
'
' Ich richte hier zuerst einen großen Puffer ein, da ich davon
' ausgehe, daß Sie sich mit der Zeit Font-Sätze unterschiedlicher
' Größe zulegen werden.
' Dieser 100 KByte-Zwischen-Puffer reicht für max.:
'   48*64    oder    32*96    oder    16*194   oder
'   64*48    oder    80*39    oder    96*33    oder
'   112*27   oder    128*24   oder    ... etc.
'   Punkte-Fonts
'
f.name$="SUPER.FON"              ! Font-Name
IF EXIST(f.name$)                ! Font-File gefunden?
  @fontload(f.name$,VARPTR(fontbuff%(0)),*platz%)! Font laden
  '
  ' Nachdem der Font durch die Lade-Prozedur geladen wurde, wird der
  ' engültig notwendige Puffer an die in Platz% zurückgegebene,
  ' tatsächliche Font-Größe angepaßt.
  '
  DIM font1%(INT(platz%/4)+1)    ! End-Puffer einrichten
  SWAP fontbuff%(),font1%()      ! Puffer-Titel austauschen
  ERASE fontbuff%()              ! Alten Puffer löschen
  f.adr1%=VARPTR(font1%(0))      ! Font-Startadresse ermitteln
  '
  ' Diese Adresse wird an die Ptext-Routine übergeben. D.h. also,
  ' daß beliebig viele Fonts gleichzeitig eingesetzt werden können,
  ' bzw. solange der Speichervorrat reicht.
  ' Für jeden Font muß ein eigenes Puffer-Feld eingerichtet werden.
  ' Das Puffer-Feld für den hier geladenen ersten Font habe ich
  ' willkürlich Font1%() genannt.
  ' Durch POKE F.adr1%,Breite und/oder POKE F.adr1%+1,Höhe
  ' können Sie zusätzlich auch die Maße der Einzel-Zeichen in
  ' relativ freien Grenzen variieren.
  '
```

```
    DEFFILL ,2,4                  !----------.
    PBOX 0,0,639,399                         !
    FOR i%=1 TO 10                           !
      a$=a$+CHR$(i%)                         !
    NEXT i%                                  ! - Irgendeinen
    FOR i%=16 TO 22                          !   String
      a$=a$+CHR$(i%)+CHR$(i%+32)             !   zusammenbasteln
    NEXT i%                                  !
    FOR i%=65 TO 72                          !
      a$=a$+CHR$(i%)+CHR$(i%+32)             !
    NEXT i%                       ! ----------'
    FOR i%=1 TO 14                ! 14 PUT-Grafikmodi
      @ptext(2,i%*28,636,a$,f.adr1%,i%) ! PTEXT-String ausgeben
    NEXT i%
  ENDIF
  '
  PROCEDURE ptext(xtxt%,ytxt%,plen%,txt$,c_bf%,gmod%)
    ' Gibt SUPER-Font-Text an beliebiger Bildschirmposition aus.
    '
    ' Xtxt%   : X-Position der linken unteren Ecke des ersten Zeichens
    ' Ytxt%   : Y-Position der linken unteren Ecke des ersten Zeichens
    ' Plen%   : Gewünschte Länge der Textausgabe vom linken Rand des
    '           ersten Zeichens zum rechten Rand des letzten Zeichens
    '           Wird Null übergeben, wird der Text in der Originallänge
    '           ausgegeben.
    ' Txt$    : Beliebiger Textausdruck
    ' C_bf%   : Startadresse des gewünschten Font-Puffers
    ' Gmod%   : Gewünschter Grafikmodus (0 - 15 siehe PUT)
    '
    ' Bei den ersten vier Parametern wurde darauf geachtet, daß ihre
    ' Übergabe-Konventionen denen des TEXT-Befehls ähnlich sind,
    ' wobei der dritte Parameter jedoch nicht optional ist und auch
    ' kein Interword-Spacing erlaubt (Minus-Länge siehe TEXT).
    '
    LOCAL i%,s_bf$,plc%,ldiv,pt_w%,pt_h%,plane%
    IF LEN(txt$)>0              ! Ist überhaupt Text vorhanden?
      pt_w%=PEEK(c_bf%)         ! Zeichenbreite lesen
      pt_h%=PEEK(c_bf%+1)       ! Zeichenhöhe lesen
      plane%=2^(2-XBIOS(4))     ! Anzahl der Bit-Planes
      plc%=((INT((pt_w%/16)+SGN(pt_w% MOD 16)))*2*pt_h%)*plane%
      ' Speicherbedarf eines Zeichen in Bytes berechnen
      s_bf$=MKI$(pt_w%-1)+MKI$(pt_h%-1)+MKI$(plane%)+SPACE$(plc%)
      '                         ! PUT-String vorbereiten
      IF plen%>0                ! Ausgabelänge größer Null?
        ldiv=plen%/LEN(txt$)+(plen%/LEN(txt$)-pt_w%)/(LEN(txt$)-1)
      '                         ! Dann Einzelzeichen-Offset berechnen
      ELSE                      ! Ausgabelänge gleich oder kleiner Null
        ldiv=pt_w%              ! dann Offset = Zeichenbreite
      ENDIF
      FOR i%=0 TO LEN(txt$)-1  ! Alle String-Zeichen...
        BMOVE c_bf%+2+ASC(MID$(txt$,i%+1,1))*plc%,VARPTR(s_bf$)+6,plc%
        '            ... aus dem Font-Puffer in den PUT-Puffer kopieren
        PUT xtxt%+i%*ldiv,ytxt%-pt_h%,s_bf$,gmod%
        '                       ! Zeichen als PUT-String putten
      NEXT i%
    ENDIF
  RETURN
  '
```

```
PROCEDURE fontload(sel$,adr%,plc%)
  ' Lädt SUPER-Font-Datei in den Puffer (ab 'Adr%).
  '
  ' Sel$: Name der Font-Datei
  ' Adr%: Startadresse des (ausreichend großen) Puffers
  ' Plc%: Pointer auf eine Integer-Rückgabevariable, die
  '       nach Abschluß die tatsächliche Größe (in Bytes)
  '       der geladenen Font-Datei enthält.
  OPEN "I",#99,sel$              ! Font-Datei öffnen
  BGET #99,adr%,LOF(#99)         ! Font in den Puffer laden
  *plc%=LOF(#99)                 ! Font-Größe ermitteln
  CLOSE #99                      ! Font-Datei schließen
RETURN
```

18. AES-Bibliothek

Diese Bibliotheks-Funktionen bietet Ihnen in der V3.0-Version unter direkt einsetzbaren Funktionsnamen die komplette AES-Bibliothek (Library).

18.1 Das Resource-Construction-Set

Bevor ich damit beginne, die einzelnen AES-Funktionen zu erläutern, muß hier erst einmal dem erfreulichen Umstand Rechnung getragen werden, daß auf der Original-GFA-V3.0-Programmdiskette ein Programm enthalten ist, das für den Einsatz von AES-Menüs und -Dialogboxen fast lebensnotwendig ist: das sog. RCS.PRG (Resource-Construction-Set).

Sicher können Resource-Files auch "per Hand" erstellt werden, was aber dermaßen kompliziert ist, daß es für etwa 99 Prozent aller Interessierten keine Möglichkeit bietet, es zu realisieren. Womit ich nicht sagen will, daß 99 Prozent aller GFA-Programmierer es nicht begreifen könnten, sondern daß 99 Prozent angesichts des gewaltigen Aufwands lieber darauf verzichten und per Programm eigene Dialogboxen und Menüs erstellen würden. Das ist nämlich tatsächlich erheblich einfacher, als ein Resource-File zu konstruieren.

Da nun aber endlich ein fehlerfreies RCS für "jedermann" zur Verfügung steht und dieses auch noch um einiges komfortabler ist, als die erste - absturzsichere (soll heißen: "mit Sicherheit abstürzende") - Version, wird die Verwendung von Resource-Files - vor allem in GFA-BASIC - einen rasanten Aufschwung erleben.

Mit einem Resource-Construction-Set ist es auf fast spielerische Art möglich, selbst die aufwendigsten Dialogboxen und Menüs zu konstruieren. Das RCS erstellt den kompletten, fertigen Objektbaum - bzw. auch mehrere Bäume parallel - und liefert drei Dateien.

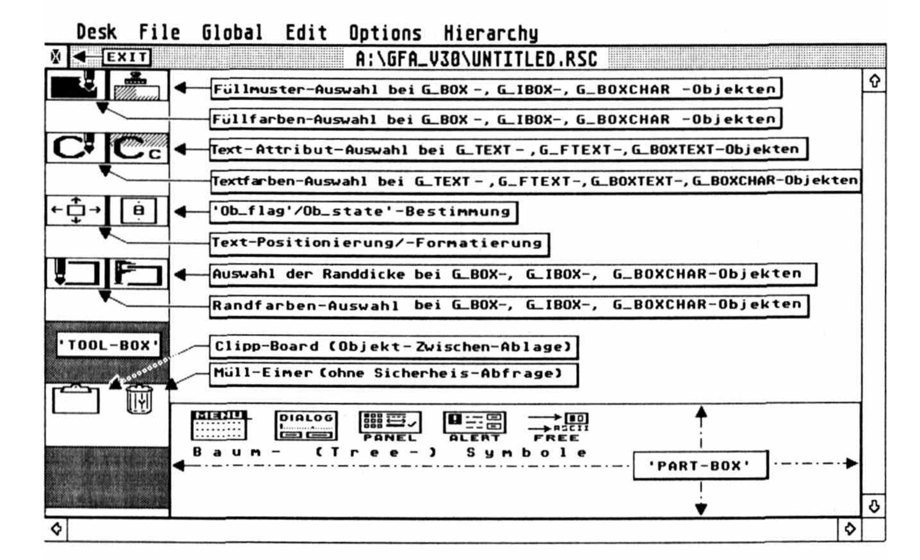
Die erste Datei ist das Resource-File selbst, in welchem sämtliche Objekt-Beschreibungen, Strukturzeiger und was sonst noch alles dazugehört, abgelegt sind. Diese Datei wird dann in unser eigenes Programm geladen (siehe RSRC_LOAD) und kann nun von dort aus eingesetzt und verwaltet werden. Diese Datei endet mit der Extension .RSC.

Die zweite Datei trägt die Extension .DFN und enthält sämtliche Daten, die für das RCS dann wichtig sind, wenn ein bereits erstelltes Resource-File hineingeladen werden soll, um es weiter zu bearbeiten. Dies sind z.B. die Baum- und Objektnamen, sowie die Baum- und Objekttypen, die Sie im RCS zugeordnet haben. Auf der Original-GFA-V3.0-Diskette finden Sie ein weiteres Programm mit Namen DEF2DFN.PRG, das die Aufgabe hat, durch das alte RCS (Version 1.0) erstellte .DEF-Dateien in die für die neue Version notwendigen .DFN-Dateien umzuwandeln.

Die dritte Datei, die vom RCS erzeugt wird, hat einen variablen Aufbau. In dieser Datei werden sämtliche, zur Weiterverarbeitung notwendigen Baum- und Objektindizes abgelegt. Da das RCS selbstverständlich nicht allein für GFA-BASIC produziert wurde, werden beim Aufbau dieser Header-Datei die für die jeweilige Programmiersprache erforderlichen Konventionen berücksichtigt.

Die RCS-V2.0-Version ermöglicht nun auch die Ausgabe GFA-spezifischer .LST-Files, die dann in ein GFA-BASIC-Programm durch Merge eingelesen werden können und durch LET allen im RCS verwendeten Baum- und Objektnamen (als Variable) den entsprechenden Baum- bzw. Objektindex zuordnen. Mit diesen Indizes sind dann innerhalb des Programms sämtliche Objekte innerhalb ihres Objektbaums ansprechbar. Welches Objekt zu welchem Baum gehört, wird als !-Kommentar der jeweiligen Deklaration angehängt.

Soviel zu den "Äußerlichkeiten", wir kommen nun zur RCS-Bedienung. Nachdem Sie das RCS2.PRG gestartet haben, meldet sich folgende Programmoberfläche:

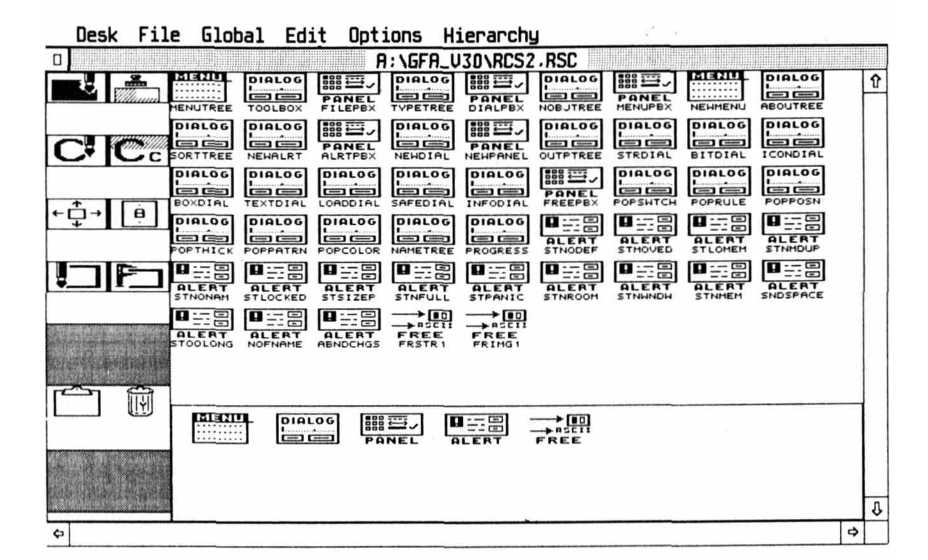


Die Programm-Optionen sind im RCS alle in englischer Sprache beschrieben, so daß es einigen von Ihnen Mühe bereiten wird, diese auf Anhieb zu verstehen. Aus diesem Grund führe ich nun alle wichtigen Optionen der Reihe nach auf und werde versuchen, sie mit kurzen Worten zu umschreiben.

Was sich auf der RCS-Oberfläche abspielt, ist - glaube ich - aus der obigen Grafik relativ leicht zu erkennen. Wundern Sie sich nicht, wenn die links in der Tool-Boxangeordneten Optionen nicht immer verfügbar sind. Sie können einen Tool-Menüpunkt immer nur dann anwählen, wenn der Menüpunkt für das aktuell aktivierte Objekt (Objekt anklicken) zulässig ist. Aber so weit sind wir noch nicht.

Wollen Sie ein bereits bestehendes Resource-File bearbeiten, so gehen Sie in das erste Pull-Down-Menü "File" und wählen dort "Open" an. Ihnen werden nun alle auf der aktuellen Diskette verfügbaren .RSC-Files angezeigt und Sie können das betreffende File laden.

Ich habe das hier mit dem RCS-eigenen .RSC-File getan.



Durch Doppelklick auf eines dieser Objektbaum-Symbole können Sie den betreffenden Objektbaum öffnen und ihn bearbeiten. Bei fremden Programm-RSCs darf dabei auf keinen Fall (!) die innere Struktur des Baumes verändert werden, da sonst die Objekt-Indizes im .RSC-File ebenfalls verändert werden und in dem fremden Programm die einzelnen Objekte mit falschen Funktionen belegt werden. Sie können allerdings die Größe, Lage und - teilweise - den Inhalt (Text) verändern, solange Sie eben darauf achten, daß die eigentliche Struktur unverändert bleibt. Das Hinzufügen von neuen Objekten ist bei fremden .RSCs also strengstens verboten, da sonst das betreffende Programm mit einem so veränderten .RSC-File nicht mehr arbeiten kann.

Damit dies alles nicht "zufällig" geschieht, werden Sie vom RCS ggfs. auf diese Veränderungen und mögliche Fehlfunktionen aufmerksam gemacht.



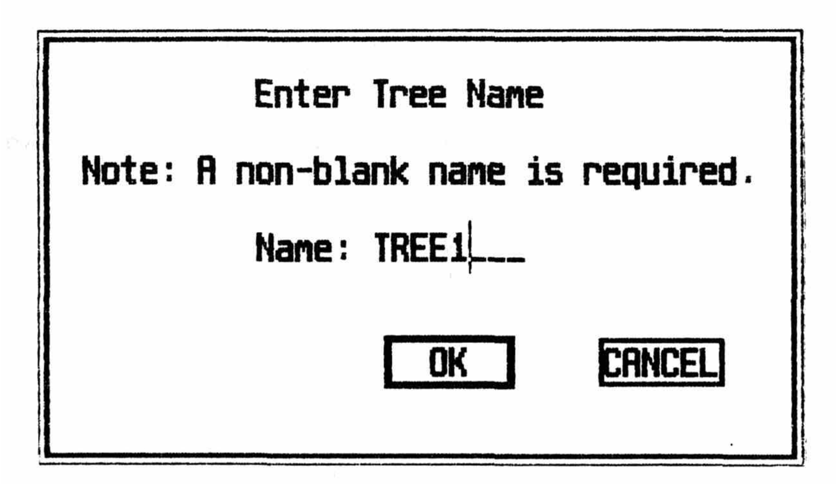








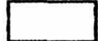
Möchten Sie ein neues .RSC-File erstellen, so suchen Sie sich zunächst aus den Baum-Symbolen das gewünschte aus, klicken es an und schieben es mit gedrückter Maustaste in das große Edit-Fenster.



Nachdem Sie es losgelassen haben, erscheint die Aufforderung, dem Baum einen passenden Namen zu geben. Tun Sie das nicht, so wird der vom RCS empfohlene Name (hier: TREE1) übernommen. Wollen Sie eigene Namen einsetzen, so achten Sie darauf, daß darin keine Spaces (Blanks, Leerzeichen) enthalten sind.

Mit einem Doppelklick auf das Baum-Symbol können Sie es nun öffnen. Je nachdem, welchen Objektbaum-Typ Sie gewählt haben, erscheinen am unteren Bildschirmrand in der Part-Box die dafür zulässigen Parts (Teile).

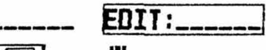
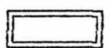
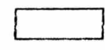
Bei Menü-Bäumen:

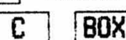
TITLE ENTRY 

Bei Alert-Bäumen:

Button Message Line   

Bei Dialog- und Paneel-Bäumen:

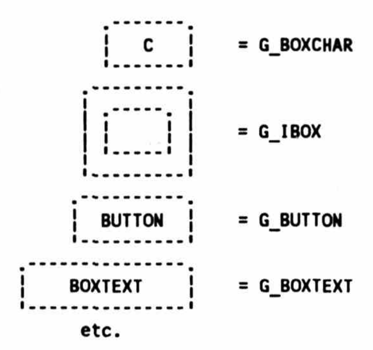
BUTTON STRING EDIT:______   

TEXT C    

und bei Free-Bäumen:

Free-string

Die darin enthaltenen Parts stellen jedes für sich einen Objekt-Typ (siehe unter OB_TYPE()) z.B.:



Diese Parts können Sie nun je nach Bedarf mit gedrückt gehaltener Maustaste in das Edit-Fenster schieben. Hier sollte allerdings - sofern vom RCS nicht schon getan - zu Anfang ein G_BOX- oder G_IBOX-Objekt als "Mutter-Objekt" angeordnet werden, in das dann die "Kinder" eingefügt werden können.

Die einzelnen Objekte (auch das Mutterobjekt) können nun per Mausklick aktiviert (rechts unten erscheint eine schwarze Box) und bearbeitet werden (Größe, Inhalt, Attribute, Farbe ect. bestimmen). In den Fällen, wo Sie das Mutterobjekt so mit "Unter-Objekten" vollgepackt haben, daß es durch Mausklick nicht mehr erreichbar ist, hilft Ihnen die <Control>-Taste weiter. Ist Sie gedrückt, wird beim einem Mausklick immer das Mutterobjekt angesprochen.

Möchten Sie einen weiteren Baum bearbeiten, schließen Sie den aktuellen Baum (Exit-Box links oben) und ziehen das neue Baumsymbol in das große Fenster. Für jeden neuen Baum gelten dieselben Erläuterungen wie für das erste.

Nachdem Sie eine komplette Resource aufgebaut haben, gehen Sie in das File-Menü und speichern es mit "Save As" ab. Evtl. sollten Sie vorher mit der Output...-Option im Global-Menü die gewünschten Index-Ausgabedateien bestimmen. Womit wir bei den Menü-Optionen wären.

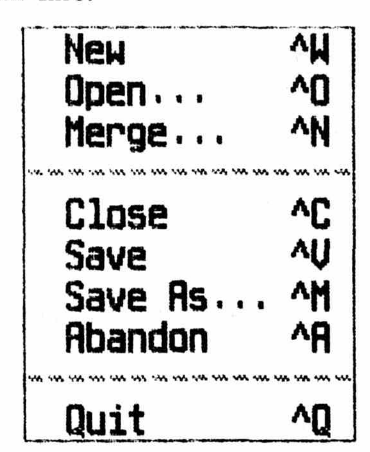
In der Menüleiste finden Sie der Reihe nach:

Desk	File	Global	Edit	Options	Hierarchy

Hier muß ich mich für eine kleine Unachtsamkeit meinerseits entschuldigen. In den folgenden Grafiken steht jeweils neben dem betreffenden Menüpunkt die Tastenkombination, durch die die Option auch über die Tastatur angewählt werden kann. Dies sind entweder <Tasten> in Verbindung mit der <Control>-Taste (^ <Taste>) oder in Verbindung mit der <Alternate>-Taste. Als Symbol für die Alternate-Taste wurde von Digital Research die schwarze Raute (ASCII=7) gewählt. Da ich jedoch bei Ausdruck der Grafiken einen anderen Zeichensatz geladen hatte (übrigens der gleiche Zeichensatz, den Sie auch auf der beiliegenden Diskette finden: SYSTEM.FNT), sehen Sie in den Grafiken anstatt der Raute zwei ineinanderliegende Rechtecke. Ich merke dies hier nur an, damit Sie nicht denken, ich hätte ein anderes RCS als Sie.

Unter den jeweiligen Menüleisten-Punkten erscheinen die folgenden Optionen:

Desk Hierunter finden Sie evtl. vorhandene Accessories und ein Programm-Info.



File Hierin sind die Ein- und Ausgabefunktionen enthalten.

New Die aktuelle Resource wird gelöscht, wenn sie geladen wurde und bisher unverändert geblieben ist. Bei neu er-

stellten oder geladenen und veränderten Resources wird vorher gefragt, ob sie gespeichert werden soll.

Open
Ein .RSC-File wird von der Diskette geladen und überschreibt die vorhanden Resource (evtl. vorher abspeichern).

Merge
Es kann ein .RSC-File von der Diskette geladen werden, das in die bestehende Resource eingefügt wird.

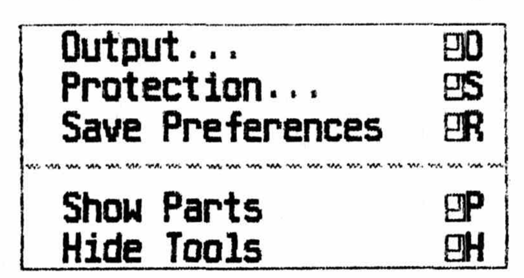
Close
Ist gerade ein Objektbaum geöffnet, wird er geschlossen. Wird "Close" auf der obersten Ebene gewählt, so hat es die gleiche Wirkung wie "New".

Save
Wurde ein Resource geladen und soll unter demselben Namen neu gespeichert werden, geschieht dies hierdurch. Es wird dabei keine Fileselect-Box aufgerufen.

Save As
Die aktuelle Resource wird nach Angabe eines neuen Dateinamens abgespeichert.

Abandon
Hat bei neu eingerichteten Resources dieselbe Wirkung wie "New", nur daß anschließend das Programm verlassen wird.

Quit
Das Programm wird nach Sicherheitsabfrage verlassen.



Global
Hier können allgemeine Einstellungen vorgenommen werden.

Output
Erlaubt die Voreinstellung verschiedener Index-Ausgaben. Wird GFA-BASIC gewählt, so werden alle Objekt- und Baum-Indizes in einem 'merge'baren .LST-File abgespeichert.

Select output files to be created:

Application binding files

- ☐ Order binding file
- ☐ 'C' (*.H)
- ☐ Pascal (*.I)
- ■ GFA-BASIC (*.LST)
- ☐ FTN-77 (*.F)

Source file for resource

- ☐ (*.RSH)

To Exit:

[OK]

[CANCEL]

Protection Es kann festgelegt werden, ob Änderungen innerhalb der Baumstruktur verhindert (LOCKED), bzw. ob vorher eine Warnung ausgegeben werden soll (NORMAL). Die dritte Möglichkeit ist, daß Änderungen an der Struktur ohne Warnung ausgeführt werden können (EXPERT), was aus weiter oben genannten Gründen dazu führen kann, daß bei Änderungen der Resource fremder Programme, diese anschließend nicht mehr korrekt verarbeitet werden können.

Choose A Resource Editing Mode:

[LOCKED] Objects may be edited, sized, or moved, but the object tree structure may not be changed.

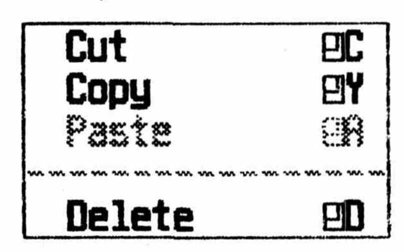
[NORMAL] A warning is given before the workspace is cleared or trees are rearranged.

[EXPERT] No warnings are given when trees are altered or the workspace is cleared.

To Exit: [OK] [CANCEL]

Save Preferences
: Die aktuell eingestellten globalen Optionen werden abgespeichert. Beim nächsten Aufruf des RCS werden diese Optionen voreingestellt.

Hide/Show Parts
Hide/Show Tools
: Die üblicherweise am unteren Bildrand eingeblendete Part-Box (Tool-Box) wird aus- bzw. wieder eingeblendet.



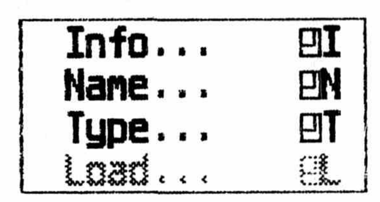
Edit — Hier können allgemeine Einstellungen vorgenommen werden.

Cut — Das aktuell aktive Objekt wird auf das Clipboard gelegt und kann von dort bei Bedarf wieder abgeholt werden.

Copy — Das aktuell aktive Objekt wird auch hier auf das Clipboard gelegt. Es bleibt dabei jedoch an seiner alten Position erhalten.

Paste — Das auf dem Clippboard liegende Objekt kann neu positioniert werden (nach Klick auf "Paste" Maustaste gedrückt halten).

Delete — Das aktuell aktive Objekt wird ohne Vorwarnung gelöscht.



Options — Dies sind auf das aktuell aktive Objekt bzw. den aktivierten Baum bezogene Optionen.

Info Liefert allgemeine Informationen zum aktuell aktiven Objekt/Baum (belegte Bytes etc.)

Information for:
A:\GFA_V30\RCS2.RSC

Objects: 589 Tedinfos: 27 Trees: 50

Iconblks: 7 Images: 48

Bitblks: 34 Strings: 470

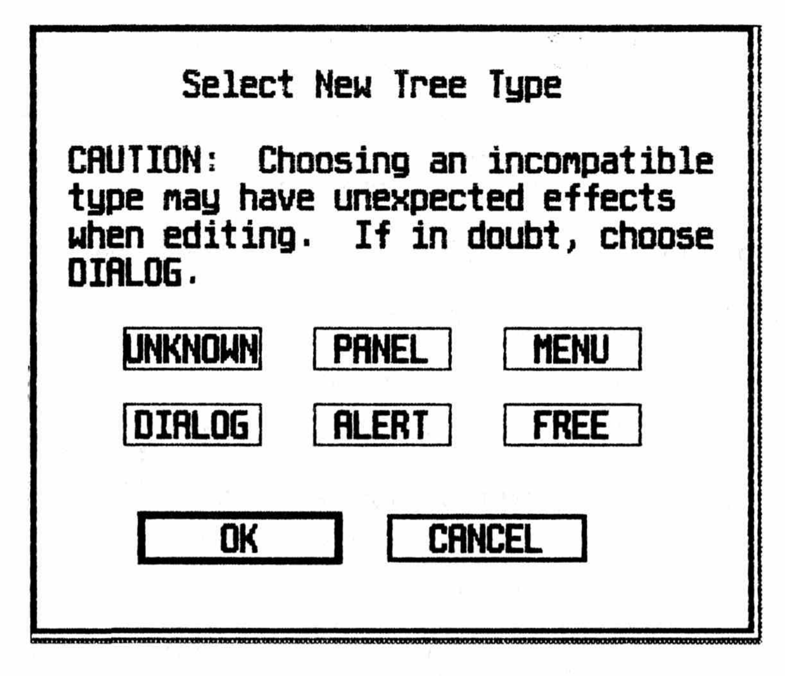
Total bytes for above: 28289

Bytes remaining in workspace: 35882

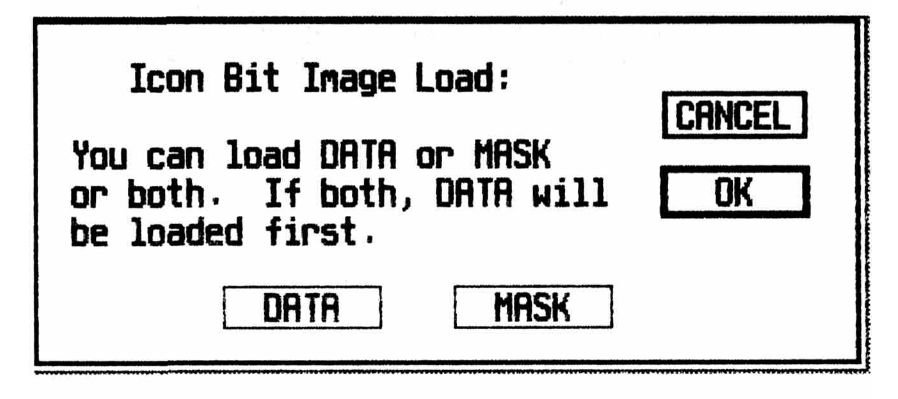
OK

Name Ermöglicht die Änderung des Objekt-/Baum-Namens. Bei Texteingabe-Objekten innerhalb des RCS dürfen keine Tiefstriche (Underscore) verwendet werden, da es dann einfach abstürzt. Dasselbe gilt übrigens auch für die Pfad-Zeile in FILESELECT-Boxen.

Type Ermöglicht die Änderung des Objekt-/Baum-Typs. Bei Objekten wird nur die jeweils mögliche Änderung angeboten. Bei Baumtyp-Änderung kann in jeden anderen Baum-Typ gewechselt werden.



Load Hiermit kann eine ICONBLK- oder BITBLK-Struktur geladen werde, falls gerade ein Grafik-Typ aktiv ist.



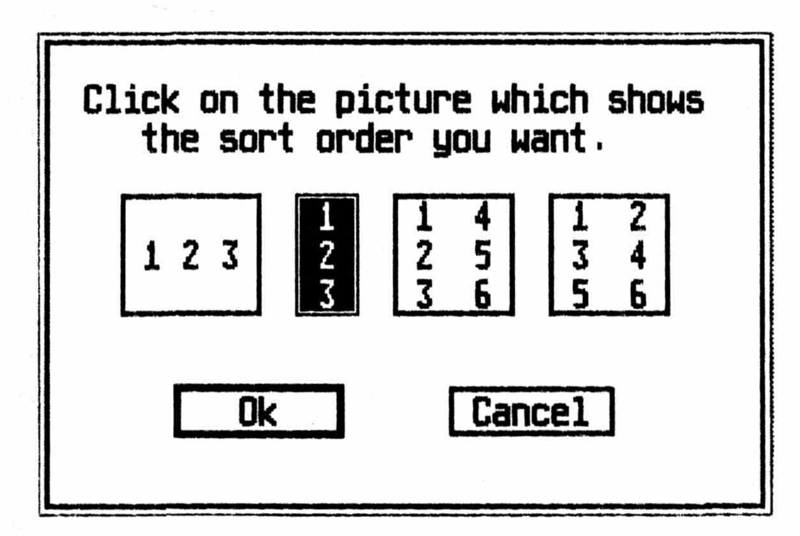
Sort Children... ^F
Unhide Children ^U

Remove Parent

Options: Index-bezogene Optionen.

Sort Children

Die aktuelle Objekt-Indizierung (normalerweise in der Reihenfolge ihrer Installation) innerhalb des betreffenden Baumes kann sortiert werden. Die Sortierfolge kann dabei vorbestimmt werden.



Unhide Children

Ist das aktuell aktive Mutterobjekt mit dem Hidden-Status (HIDETREE siehe OB_FLAGS()) versehen, wird der Status hierdurch wieder aufgehoben.

Remove Parent

Diese Funktion hat mir ihr Geheimnis leider noch nicht offenbart.

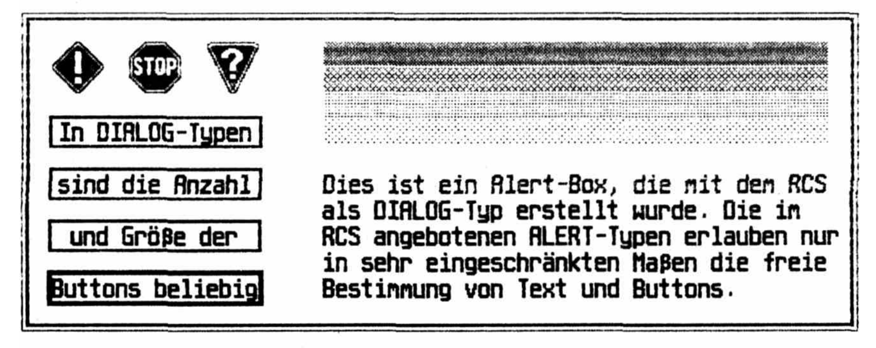
Als kleine Übung soll hier die Erstellung und der Einsatz einer Alert-Box demonstriert werden.

Im Normalfall besteht eine Alert-Box aus einem Image (G_IMAGE/BITBLK), einem oder mehreren Strings (G_STRING) und einem oder mehreren Buttons (G_BOXTEXT), die alle zusammen in einem Mutter-Objekt untergebracht sind (weiße Hintergrundbox).

Das Image und die Strings können nicht angewählt werden, bzw. Sie können es anklicken, aber das bleibt ohne Wirkung. Diese beiden Ob-

jekttypen haben also in einer Alert-Box keine SELECTABLE-Eigenschaft (siehe OB_FLAGS()). Die einzigen wählbaren Objekte in einer Alert-Box sind die Buttons, wovon einer ggfs. stark umrandet ist. Diese starke Umrandung bedeutet, daß dieser Button auch durch Druck auf die <Return>-Taste gewählt werden kann. Die normal umrandeten Buttons haben also zusätzlich zur SELECTABLE-Eigenschaft "nur" die EXIT-Eigenschaft. Der dick umrandete Button hat dagegen drei Eigenschaften gleichzeitig. Er ist erstens SELECTABLE (wählbar), zweitens kann durch ihn mit einem Mausklick das Formular verlassen werden (EXIT) und drittens ist es ein DEFAULT-Objekt, daß auch mit der <Return>-Taste gewählt werden kann (siehe OB_FLAGS()). Außerdem wurde ihm in OB_STATE() der grafische Status OUTLINED zugewiesen.

Da das vom RCS vorgegebene ALERT-Format nur sehr bedingt brauchbar ist, habe ich eine Alert-Box im DIALOG-Format erstellt, was ohne weiteres genauso einfach ist, wie die Erstellung im ALERT-Format.



Wenn Sie eine solche Alert-Box im RCS erstellen wollen, holen Sie ein Alert-Symbol ins große Fenster. Nachdem Sie es durch Doppelklick geöffnet haben, finden Sie schon eine Alert-Box vorbereitet.

Um ein ALERT-Symbol (Ausrufungszeichen, Stop-Schild, Fragezeichen) in einen DIALOG-Baum zu transportieren, wählen Sie aus der Alert-Part-Box das gewünschte Symbol, holen es in das Mutterobjekt und schieben es dann auf das Clipboard. Nun können Sie die Alert-Box schließen und in den Mülleimer werfen. Nachdem Sie ein DIALOG-Symbol geöffnet haben, holen Sie das ALERT-Symbol vom "Klemmbrett" und plazieren es im DIALOG-Mutterobjekt. Wie Sie an der untenstehenden Grafik sehen, läßt sich das auch mit mehreren Symbolen machen. Da jedoch das Klemmbrett immer nur ein Objekt

aufbewahrt, müssen Sie dazu mehrmals zwischen der ALERT- und der DIALOG-Box hin- und herwechseln.

Nun plazieren Sie in der DIALOG-Box alle Objekte, die Sie für nötig halten. Dabei wird den Objekten der Objekt-Index in der gleichen Reihenfolge zugeteilt, in der Sie die Objekte in das Formular einbauen. Durch die Sort Children-Option können Sie diese Reihenfolge ordnen.

Ich habe diesen Vorgang für Sie einmal durchgespielt und eine ziemlich freie Alert-Box erstellt. Wie Sie sehen, ist in einer Dialog-Alert-Box keine Begrenzung des Textes, der Button-Länge oder der Objektanzahl vorgegeben. Die Verwaltung einer solchen Box ist nun in GFA-V3.0 denkbar einfach.

```
LET alert&=0      ! RSC_TREE (Objektbaum)  ---.
LET button1&=7    ! Objekt in Baum 0          | Dies sind die Objekt-
LET button2&=9    ! Objekt in Baum 0          | Indizes, die Ihnen
LET button3&=12   ! Objekt in Baum 0          | das RCS liefert.
LET default&=15   ! Objekt in Baum 0       ---'
'
' Ich habe hier die Indizes der Nicht-SELECTABLE-Objekte
' (Strings, Balken und Images) herausgenommen, weil uns hier
' nur die Objekte interessieren, durch die das Formular
' verlassen werden kann. Es ist natürlich auch denkbar, daß
' die Objekte die Eigenschaft SELECTABLE bekommen,
' aber nicht dazu verwendet werden, das Formular zu verlassen. Sie
' dürfen dann nicht die TOUCHEXIT-, EXIT-, oder DEFAULT-
' Eigenschaft haben.
' Ob solche Objekte vor Formular-Exit angewählt (invertiert) wurden,
' läßt sich dann anschließend feststellen, indem man das
' SELECTED-Bit in ihrem OB_STATE() abfragt.
' Am Rande bemerkt: es ist pro Formular
' (logischerweise) nur ein DEFAULT-Objekt möglich.
'
RESERVE FRE(0)-10000              ! Speicher für Resource freimachen
okay%=RSRC_LOAD("alert.rsc")      ! Resource laden
IF okay%                          ! Ladevorgang ohne Fehler?
  ~RSRC_GADDR(0,0,baum%)          ! Baum-Startadresse holen
  ~FORM_CENTER(baum%,xl%,yl%,br%,ho%) ! Koordinaten zentrieren
  ~OBJC_DRAW(baum%,0,1,xl%,yl%,br%,ho%) ! Baum zeichnen
  exit%=FORM_DO(baum%,0)          ! Formular bedienen
  IF exit%=15                     ! DEFAULT-Button gewählt
    PRINT "DEFAULT-Button gewählt!"
  ELSE                            ! Anderer Button
    PRINT "Index des gewählten Buttons: ";exit%
  ENDIF
  ~RSRC_FREE()                    ! Resource löschen
ELSE
  ALERT 1,"'ALERT.RSC' nicht gefunden !",1,"Return",b%
ENDIF
RESERVE FRE(0)+10000              ! Speicher restaurieren
```

Es ist übrigens sehr wichtig, darauf zu achten, daß mindestens ein EXIT-Objekt im Formular enthalten ist, da Sie sonst Ihren ST neu starten müssen. Ein GEM-Formular kann auch durch <Control>-<Shift><Alternate> nicht verlassen werden!

Bezüglich der im RCS verwendeten freien Icons und Images gibt es ein Problem. Haben Sie im RCS ein Image-(BITBLK)- oder Icon-(ICNBLK)-Objekt aktiviert, kann über die Load-Option im Options-Menü ein Image oder Icon von der Diskette (Extension: .ICN) geladen werden. Dies ist eine sehr nützliche Option, wenn man weiß, in welchem Format die Bilder erwartet werden. Ich habe (nicht übertrieben!) einen ganzen Tag damit verbracht, alle möglichen Icon-Formate zusammenzubasteln und sie dem RCS anzubieten. Der einzige Effekt des ganzen Aufwands war, daß das Programm jedesmal in aller Seelenruhe abstürzte. Am nächsten Tag war ich dann ca. 8 Stunden damit beschäftigt, halb Deutschland anzurufen (ca. 20 - zum Teil hochkarätige - ST-Kenner). Aber es wußte einfach keiner, wie das gemacht wird. Die einzige Auskunft, die ich erhielt, bestand darin, doch einfach das Image/Icon mit einem Icon-Editor zu produzieren, es von diesem abspeichern zu lassen und dann zu laden. Schön und gut - nur was tue ich, wenn ich gerade keinen Icon-Editor habe, der das vom RCS gewünschte Format erzeugt?

Ich weiß es bis heute nicht! Als ein Ausweg aus diesem Problem bleibt der Weg über einen RCS-kompatiblen Icon-Editor. Einen solchen finden Sie z.B. im Entwicklungspaket von Digital Research. Eine weitere Möglichkeit ist, mit dem RCS ein Mutterobjekt zu erzeugen, ein Image hineinzupacken, diesen Baum abzuspeichern und anschließend das Dummy-Image gegen das eigene per BASIC auszutauschen (z.B. durch eine DATA/READ-Schleife die Bit-Muster hineinpoken). Wer zufällig das MEGAMAX-RCP (Resource-Construction-Programm) zur Hand hat, kann sich bei ICNBLK-Icons diese Arbeit erleichtern, indem er ein Icon mit dem dort integrierten Icon-Editor bearbeitet, es in ein Mutter-Objekt packt und das Ganze als alleinstehendes RSC-File abspeichert.

Ein so erzeugtes .RSC-File können Sie nun in das RCS "mergen". Öffnen Sie den Baum und legen Sie das Icon auf dem Klemmbrett ab. Den nun überflüssigen Icon-Mutterbaum können Sie jetzt löschen und das Icon vom Klemmbrett in einen beliebigen anderen Baum einfügen.

Ich hoffe, daß Ihnen diese Kurz-Einführung in die Arbeit mit dem Resource-Construction-Set am Anfang den Umgang mit den vielfältigen Optionen etwas erleichtern wird. Mit einiger Übung und den In-

formationen zur Verwaltung einer GEM-Resource im nächsten Kapitel sind Sie dann in der glücklichen Lage, sich die Programmierung komplexer Abfrage- und Auswahl-Strukturen durch GEM-Menüs und -Dialogboxen wesentlich zu erleichtern.

18.2 Vorbemerkungen zu den AES-Funktionen

Da es sich bei der AES-Library ca. 70 teilweise sehr komplexe Funktionen handelt, hätte es den Rahmen des Buches gesprengt, hier eine komplette V2.xx-AES-Library mit allen dazugehörigen GCONTRL/GINTIN/ADDRIN-Belegungen und den entsprechenden GEMSYS-Aufrufen vorzustellen. Ich bin der Meinung, daß dazu auch zwischenzeitlich schon soviel qualifizierte und ausführliche Lektüre erschienen ist (auch fertige V2.xx-Libraries), daß hier mit einer nochmaligen Behandlung des Themas des Guten zu viel getan wäre.

Die Form der V3.0-Funktionsaufrufe ist der in C üblichen Art und Weise angepaßt. "Überläufer" von C zu BASIC werden sich aufgrund dieser Tatsache recht schnell mit diesen Funktionen zurechtfinden.

Bei den nun folgenden Funktionsbeschreibungen sind bei der Angabe der Aufruf-Syntax **drei Besonderheiten** zu beachten. Die erste ist die, daß als Syntaxform grundsätzlich nur die Form der Zuweisung verwendet wurde, obwohl Funktionen auch durch VOID, durch Direkteinbindung (z.B. A$="***"+STR$(APPL_INIT())+"***") oder in Print-Befehlen aufrufbar sind (z.B. PRINT "Rückgabe: ";APPL_INIT()).

Die zweite Besonderheit ist, daß Rückgabeparameter (ggfs. auch optional), welche die AES-Rückgabewerte intern aus GINTOUT und ADDROUT aufzunehmen haben, hier durch einen rückwärtsweisenden Doppelpfeil vor dem Variablennamen gekennzeichnet sind (z.B. <<Back1,<<Back2,...). Das geschieht, um Mißverständnisse bei der Ergebnisauswertung zu vermeiden. Diese Doppelpfeile haben in der regulären BASIC-Syntax **nichts** zu suchen.

Als dritte Besonderheit ist zu erwähnen, daß die Syntax vieler dieser Funktionen nicht in einer einzelnen Druckzeile unterzubringen sind. In diesen Fällen wird die Zeilentrennung - wie im übrigen Buch auch - durch je drei Punkte (...) kenntlich gemacht. Ebenso wie die erwähnten Doppelpfeile dürfen auch diese Punkte in der BASIC-Syntax nicht verwendet werden.

Bei den Funktions-Parametern, in denen Adressen zu übergeben sind, bzw. in den Adressen zurückgegegben werden, ist zu beachten, daß dazu grundsätzlich 4-Byte-Integervariablen, sowie bei Koordinaten- und übrigen Werteparametern mindestens 2-Byte-Integervariablen verwendet werden.

Außerdem wird im folgenden häufig der Begriff "Baum" verwendet. Dieser Ausdruck ist als Synonym für die Gesamtliste aller zu einem Formular (Objektbaum/Menübaum) gehöriger Objekte zu verstehen. Seine Startadresse ist die Adresse des ersten (Wurzel-)Objektes dieser Liste.

Beispiele zu den einzelnen Funktionen finden Sie ggfs. in Kurzform unter der jeweiligen Funktionsbeschreibung. Zwei größere Beispiele zur Dialogbox-Verwaltung sowie zum Menü-Handling finden Sie am Ende der AES-Library.

18.3 Application-Manager

APPL_EXIT() — GEM-Applikation abmelden

Back=APPL_EXIT()

Meldet das auslösende Programm beim AES ab und gibt die ihm zugeordnete Identifikationskennziffer (ap_id) wieder frei (in GFA unnötig, siehe unter APPL_INIT()).

Back	**0 = Fehler; <> 0 = Funktion korrekt ausgeführt.**

APPL_FIND() — Applikations-Identifikator ermitteln

Back=APPL_FIND(Programmname)

GEM ist in der Lage, mehrere Programme parallel im Speicher zu "fahren". Diese Funktion ermittelt die Applikationskennziffer (ap_id) einer bestimmten Parallel-Applikation.

Back:

-1	**Angegebenes Programm im Speicher nicht gefunden.**
<> -1	**Im Erfolgsfall Kennziffer (ap_id) der gefundenen Applikation.**

Programmname:
8 Zeichen langer Programmname der gesuchten Applikation, ohne Extension. Es sind unbedingt 8 Zeichen anzugeben. Wenn nötig, ist der Name mit Leerzeichen auf 8 Zeichen zu erweitern. Bei mehr als 8 Zeichen besteht Absturzgefahr (fatal).

APPL_INIT() — GEM-Applikation anmelden

Back=APPL_INIT()

Eigentlich Dummy-Aufruf der Funktion. Die Funktion wird tatsächlich nicht ausgeführt, da der Interpreter und Compilate dies bei Start automatisch tun. Es wird lediglich die bei GFA-Start erhaltene Identifikationsnummer (ap_id -> GFA-Handle) in Back zurückgegeben. Dieses Handle wird bei verschiedenen anderen AES-Funktionen benötigt.

APPL_READ() — Aus Ereignispuffer lesen

Back=APPL_READ(Handle,Länge,Adresse)

Liest eine vorgegebene Anzahl Bytes aus einem bestimmten AES-Message-Buffer.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle (ap_id) der Parallel-Applikation, deren Puffer ausgelesen werden soll.
Länge	Anzahl Bytes, die gelesen werden sollen.
Adresse	Adresse eines Puffers, in dem die gelesenen Bytes abgelegt werden sollen (minimale Puffergröße = Länge).

APPL_TPLAY() — Wiedergabe gespeicherter Ereignisse

Back=APPL_TPLAY(Adresse,Anzahl,Tempo)

- Nur mit Blitter-TOS einsetzbar! -

Spielt das mit APPL_TRECORD aufgenommene Software-Band ab. Wiederholt also alle dort gespeicherten Ereignisse.

Back	Ohne Bedeutung (immer 1).
Adresse	Startadresse des Speicherbereichs, in dem die Ereignisfolge von APPL_TRECORD gespeichert wurde.
Anzahl	Anzahl der wiederzugebenden Ereignisse.
Tempo	Legt die Ablaufgeschwindigkeit der Wiedergabe fest: 50 = halbe ; 100 = normale ; 200 = doppelte Geschw.

Diese Funktion wurde von Digital Research fehlerhaft implementiert und ist daher kaum brauchbar. Die Ergebnisse entsprechen nicht der GEM-Dokumentation.

APPL_TRECORD() **Ereignisse speichern**

Back=APPL_TRECORD(Adresse,Anzahl)

- Nur mit Blitter-TOS einsetzbar! -

Veranlaßt das "Mitlaufen" eines Software-Bandes. Speichert also eine bestimmte Anzahl an Benutzer-Ereignissen (Tastendruck, Timer, Mausknopfbetätigung und Mausbewegung) in einem Ereignis-Puffer.

Back Anzahl der tatsächlich gespeicherten Ereignisse.

Adresse Startadresse eines Speicherbereichs, in dem die Ereignisfolge gespeichert werden soll. Puffergröße: 6 * Anzahl.

Anzahl Gewünschte Ereignisanzahl, die "mitgeschnitten" werden soll. Für jedes gespeicherte Ereignis werden 6 Byte Speicherplatz benötigt.
- <- |Byte 1|Byte 2|Byte 3|Byte 4|Byte 5|Byte 6| -> +

Byte 1 Immer 0.

Byte 2 Ereignis-Flag (Art des stattgefundenen Ereignisses):
0 = Timer-Ereignis 1 = Maustasten-Ereignis
2 = Maus-Ereignis 3 = Tastatur-Ereignis

Wenn Byte 2 = 0 (Timer):

Byte 3-6 Verstrichene Zeit in Millisekunden

Wenn Byte 2 = 1 (Maustaste):

Byte 3 0

Byte 4 Anzahl der getätigten Klicks

Byte 5 0

Byte 6 Maustastenstatus
0 = Taste nicht gedrückt 1 = Taste gedrückt

Wenn Byte 2 = 2 (Mausbewegung):

Byte 3-4 Maus-X-Koordinate zum Zeitpunkt des Ereignisses.

Byte 5-6 Maus-Y-Koordinate zum Zeitpunkt des Ereignisses.

Wenn Byte 2 = 3 (Tastatur):

Byte 3 0

Byte 4 Umschaltasten-Status:
Bit 0 = Shift rechts Bit 1 = Shift links
Bit 2 = Control Bit 3 = Alternate

Byte 5 0

Byte 6 ASCII-Code der gedrückten Taste

Diese Funktion wurde von Digital Research fehlerhaft implementiert und ist daher kaum brauchbar. Die Ergebnisse entsprechen nicht der GEM-Dokumentation.

APPL_WRITE() — In Ereignispuffer schreiben

Back=APPL_WRITE(Handle,Länge,Adresse)

Schreibt eine vorgegebene Anzahl Bytes in einen bestimmten AES-Message-Buffer.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle (ap_id) der Parallel-Applikation, in deren Puffer geschrieben werden soll.
Länge	Anzahl Bytes, die geschrieben werden sollen.
Adresse	Adresse eines Puffers, in dem die zu schreibenden Bytes liegen.

18.4 Event-Manager

EVNT_BUTTON() — Auf Maustasten-Ereignis warten

Back=EVNT_BUTTON(Klicks,Taste,Status...
...[,< <Mausx,< <Mausy,< <Mausk,< <Switch])

Stoppt den Programmlauf und wartet auf Betätigung einer Maustaste (vgl. ON MENU BUTTON..GOSUB).

Back	Anzahl der tatsächlich ausgeführten Maustastenklicks.
Klicks	Anzahl der gewünschten Maustastenklicks.
Taste	Gewünschte Maustaste, die das Ereignis auslösen soll: 1 = Linke ; 2 = Rechte ; 3 = Beide Maustasten
Status	Status von Taste: 0 = Ereignis gilt sofort als ausgelöst. 1 = Ereignis ausgelöst, wenn linke Taste gedrückt. 2 = Ereignis ausgelöst, wenn rechte Taste gedrückt. 3 = Ereignis ausgelöst, wenn beide Tasten gedrückt.

<< Optional anzugebende Rückgabe-Variablen:

Mausx	Rückgabe der Maus-X-Koordinate bei Ereignis.
Mausy	Rückgabe der Maus-Y-Koordinate bei Ereignis.
Mausk	Rückgabe des Maustasten-Status bei Ereignis (siehe Taste).
Switch	Rückgabe des Umschalttasten-Status bei Ereignis: 0 = Keine Taste gedrückt 1 = Shift rechts 2 = Shift links 4 = Control 8 = Alternate (z.B. 12 = <Control>+<Alternate>, 3 = Beide Shift-Tasten)

EVNT_DCLICK() Doppelklick-Geschwindigkeit einstellen

Back=EVNT_DCLICK(Dklick,Modus)

Bestimmt oder ermittelt einen Zeitwert, der vom GEM bei Mausklicks gewartet wird. Wird innerhalb dieser Spanne eine Maustaste zweimal gedrückt, wird dies als Doppelklick angenommen. Diese Routine verzögert bzw. beschleunigt auch die Arbeitsgeschwindigkeit bei Einfachklicks, da dem Benutzer jeweils die eingestellten Spanne zum evtl. zweiten Drücken der Maustaste Zeit gelassen wird.

Back	Alte Klick-Zeitspanne (0 = langsam ... 4 = schnell).
Dklick	Wenn Modus = 1, dann neuer Zeitwert (siehe Back). Bei Modus = 0 bleibt dieser Parameter unberücksichtigt.
Modus	Funktionsmodus: 0 = Aktuellen Zeitwert liefern. 1 = Den in Dklick angegebenen Zeitwert einstellen.

EVNT_KEYBD() Auf Tastatur-Ereignis warten

Back=EVNT_KEYBD()

Stoppt Programmlauf und wartet auf Betätigung einer Tastaturtaste.

Back	2-Byte-Wert: Lobyte = ASCII-Code der gedrückten Taste. Hibyte = Scan-Code der gedrückten Taste.

EVNT_MESAG() Auf Eventpuffer-Ereignis warten

Back=EVNT_MESAG(Adresse)

Stoppt den Programmlauf und wartet darauf, daß ein Window- oder Menü-Ereignis im Ereignispuffer eingetragen wird.

Back	Ohne Bedeutung (immer 1).
Adresse	Startadresse eines 16 Byte (bzw. 8 Word) großen Puffers, der für diese Funktion als Ereignispuffer angesehen wird.

Vorsicht: Ist kein Menü oder Window installiert, das ein Ereignis auslösen könnte, wartet die Funktion unendlich.

GFA installiert grundsätzlich einen Ereignispuffer, der durch einen ON MENU-Aufruf über Interrupt aktualisiert wird. Dieser GFA-Puffer wird durch MENU(1) bis MENU(8) (8 Words) repräsentiert.

Die Einträge im EVNT_MESAG-Puffer entsprechen den Einträgen im MENU()-Feld.

EVNT_MOUSE() **Auf Mauspositions-Ereignis warten**

Back=EVNT_MOUSE(Flag,Xpos,Ypos,Breite,Höhe...
...[,<<Mausx,<<Mausy,<<Mausk,<<Switch])

Stoppt den Programmlauf und wartet darauf, daß der Mauszeiger ein bestimmtes Bildschirm-Rechteck betritt oder verläßt (vgl. ON MENU IBOX/OBOX GOSUB).

Back	Ohne Bedeutung (immer 1).
Flag	Gewünschter Modus, der das Ereignis auslösen soll: 0 = Betreten 1 = Verlassen des Rechtecks.
Xpos	X-Koordinate der linken oberen Ecke des Rechtecks.
Ypos	Y-Koordinate der linken oberen Ecke des Rechtecks.
Breite	Breite des Rechtecks in Pixeln.
Höhe	Höhe des Rechtecks in Pixeln.

<< Die optional anzugebenden Rückgabe-Variablen Mausx, Mausy, Mausk und Switch haben die gleiche Bedeutung wie bei EVNT_BUTTON.

EVNT_MULTI() **Auf Mehrfach-Ereignis warten**

Back=EVNT_MULTI(Modus,Klicks,Taste,Status,Bflag1,Xpos1,Ypos1,...
...Breite1,Höhe1,Bflag2,Xpos2,Ypos2,Breite2,Höhe2,...
...Adresse,Zeit [,<<Mausx,<<Mausy,<<Mausk,<<Switch,<<Bklicks])

Stoppt den Programmlauf und wartet auf das Eintreten eines oder mehrerer verschiedener Ereignisse. Diese Funktion stellt eine Zusammenfassung der vorher beschriebenen EVNT_Funktionen dar.

Im Gegensatz zu EVNT_MOUSE können hier zwei Rechtecke gleichzeitig überwacht werden. Diese Funktion ist vergleichbar mit dem BASIC-Befehl ON MENU, der jedoch nicht auf ein Ereignis wartet, sondern das Warten nach (sehr) kurzer Zeit abbricht und ins BASIC zurückkehrt.

Back	Tatsächlich stattgefundenes Ereignis (6-Bit-Vektor):	
	Bit 0 = an -> Tastatur-Ereignis	(EVNT_KEYBD)
	Bit 1 = an -> Mausknopf-Ereignis	(EVNT_BUTTON)
	Bit 2 = an -> Maus-Ereignis/Box 1	(EVNT_MOUSE)
	Bit 3 = an -> Maus-Ereignis/Box 2	(EVNT_MOUSE)
	Bit 4 = an -> Event-Puffer-Ereignis	(EVNT_MESAG)
	Bit 0 = an -> Timer-Ereignis	(EVNT_TIMER)

Modus	**Gibt als 6-Bit-Vektor an, auf welches Ereignis gewartet werden soll. Bit-Bedeutung wie bei Back**
Klicks	**Siehe EVNT_BUTTON.**
Taste	**Siehe EVNT_BUTTON.**
Status	**Siehe EVNT_BUTTON.**
Bflag1, Xpos1, Ypos1, Breite1, Höhe1	**siehe EVNT_MOUSE.**
Bflag2, Xpos2, Ypos2, Breite2, Höhe2	**siehe EVNT_MOUSE.**
Adresse	**Siehe EVNT_MESAG.**
Zeit	**Siehe EVNT_TIMER.**

<< Die optional anzugebenden Rückgabe-Variablen Mausx, Mausy, Mausk und Switch haben dieselbe Bedeutung wie bei EVNT_BUTTON. Die optionale Rückgabe-Variable Bklicks liefert die Anzahl der tatsächlich ausgeführten Maustastenbetätigungen.

EVNT_TIMER() — Auf Zeit-Ereignis warten

Back=EVNT_TIMER(Zeit)

Stoppt den Programmlauf und wartet, bis eine bestimmte Zeit verstrichen ist (siehe auch DELAY und PAUSE).

Back	**Ohne Bedeutung (immer 1).**
Zeit	**Zu wartende Zeitspanne in 1/1000 Sekunden.**

18.5 Formular-Manager

FORM_ALERT() — Hinweis-Formular (Alert-Box) erzeugen

Back=FORM_ALERT(Button,Text$)

Es wird eine Alert-Box erzeugt (vgl. ALERT).

Back	**Nummer des angeklickten Auswahl-Buttons.**
Button	**Nummer des Buttons (0 - 3), der auch durch <Return> gewählt werden kann (dick umrandet -> Default).**
Text$	**Format-String, der den Inhalt der Alert-Box darstellt: [Icon][Zeile1|Zeile2|...][Button1|Button2|...] Die einzelnen Komponenten sind in eckige Klammern einzufassen. Icon: Grafiksymbol, das in der linken oberen Ecke der Box angezeigt werden soll: 0 = Kein Icon 1 = Ausrufezeichen 2 = Fragezeichen 3 = Stopzeichen Zeile: Maximal 5 Textzeilen mit maximal je 30 Zeichen. Die Zeilen sind voneinander durch | (Pipe) zu trennen (vgl. ALERT).**

Button: Maximal 3 Button-Zeilen mit maximal je 10 Zeichen. Die Buttons sind voneinander durch | (Pipe) zu trennen (vgl. ALERT).

Textbeispiel:

```
Text$="[1][Dies ist|ein Alert-Box-|Beipiel!][ Na |toll]"
```

FORM_BUTTON() Mausüberwachung im Formular

Back=FORM_BUTTON(Adresse,Objekt,Klicks,<<N_objc)

- Von FORM_DO verwendete Unterroutine! -

Behandelt formularbezogene Mauseingaben. Siehe Erläuterung zu FORM_KEYBD.

Back	0 = Exit-Objekt (EXIT bzw. TOUCHEXIT) wurde gewählt. > 0 = Formular-Dialog ist noch nicht beendet.
Adresse	Adresse des bearbeiteten Objektbaumes (Formulars).
Objekt1	Nummer des aktuellen Formular-Objektes.
Klicks	Anzahl der Maustasten-Klicks.
N_objc	Rückgabevariable, die nach Abschluß die Nummer des für den nächsten Aufruf aktualisierten Formular-Objektes enthält.

FORM_CENTER() Formularkoordinaten zentrieren

Back=FORM_CENTER(Adresse,<<Xpos,<<Ypos,<<Breite,<<Höhe)

Berechnet die Koordinaten eines Formulars so, daß es durch OBJC_DRAW in Bildschirmmitte dargestellt werden kann.

Back	Ohne Bedeutung (immer 1).
Adresse	Adresse des Objektbaumes, dessen zentrierte Position berechnet werden soll.

<< Die Rückgabe-Variablen Xpos, Ypos, Breite und Höhe enthalten die zentrierten X- und Y-Koordinaten, sowie Breite und Höhe des berechneten Formulars.

FORM_DIAL() Formular-Hintergrund puffern/restaurieren

Back=FORM_DIAL(Modus,Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)

Der Zweck dieser Funktion besteht im Wesentlichen darin, den Bildschirmhintergrund bei Aufruf eines Formulars zu reservieren und bei Verlassen des Formulars wieder zu restaurieren. Dabei ist bei geöff-

neten Windows der Programmierer für die Restauration der Fensterinhalte selbst verantwortlich (Redraw). Von FORM_DIAL werden nur die Randbereiche der überlagerten Windows neu gezeichnet. Die Fensterinhalte bleiben von den Formular-"Resten" belegt. Bereiche außerhalb von Windows werden mit dem Füllmuster des Desktop-Hintergrunds (DEFFILL 1,2,4) gefüllt.

Als Effekt kann mit den Formularkoordinaten auch eine GROWBOX oder SHRINKBOX erzeugt werden.

Back — 0 = Fehler; <> 0 = Funktion korrekt ausgeführt.

Modus — Funktionsmodus:
0 = Bildschirmrechteck Bigx,Bigy,Bigb,Bigh speichern.
1 = GROWBOX produzieren (vgl. GRAF_GROWBOX).
2 = SHRINKBOX produzieren (vgl. GRAF_SHRINKBOX).
3 = Bildschirmrechteck Bigx,Bigy,Bigb,Bigh restaurieren.

Litx, Lity, Litb, Lith:
Koordinaten und Maße des kleineren Rechtecks bei Modus 1 oder 2. Werden hier die Parameter Litb und Lith mit 0 belegt, wird der Eindruck erzeugt, daß das Zentrum des größeren Rechtecks die Startkoordinate des kleineren Rechtecks ist. Bei Modus 0 oder 3 werden alle vier Parameter ignoriert.

Bigx, Bigy, Bigb, Bigh:
Koordinaten und Maße des größeren Rechtecks bei Modus 1 oder 2 bzw. Koordinaten und Maße des Hintergrunds bei Modus 0 oder 3. Im letzteren Fall können hier die Rückgabewerte von FORM_CENTER als Koordinaten und Maße eingesetzt werden.

FORM_DO() — GEM übernimmt Formular-Verwaltung

Back=FORM_DO(Adresse,Objekt1)

Das GEM verwaltet bei Einsatz dieser Funktion eine komplette Formularbedienung selbstständig. Die Bediener-Überwachung wird beendet, wenn ein Objekt vom Typ Exit oder Touchexit angeklickt wurde.

Vorsicht: Ist kein solches Objekt im Formular enthalten, gibt es keine Möglichkeit, zum BASIC zurückzukehren, da die gesamte Bedienerüberwachung sich ausschließlich auf das Formular beschränkt. Es sind weder die GFA-Abbruchfunktion <Control/Shift/Alternate>, noch eventuelle Pull-Down-Menüs im Hintergrund verfügbar.

Back — Nummer (Index) des Exit-Objektes, das als Formular-Ausgang angeklickt wurde. Wurde dazu ein Doppelklick (siehe EVNT_DCLICK) verwendet, ist das Bit 15 gesetzt (IF Back AND 2^15).

Adresse — Adresse des Objektbaumes, der dargestellt werden soll.

Objekt1 — Nummer des ersten editierbaren Textobjektes des Formulars, auf welches der Text-Cursor gesetzt werden soll. Ist kein solches Objekt vorhanden, ist hier 0 anzugeben.

FORM_ERROR() — Hinweis-Formular (TOS-Error) erzeugen

Back=FORM_ERROR(Code)

Erzeugt eine Error-Alert-Box mit dem Format:

```
[3][TOS Fehler #][Abbruch]
```

Back	Ohne Bedeutung (immer 0).
Code	Anzuzeigender Fehler-Code.

FORM_KEYBD() — Formular-Texteingabe

Back=FORM_KEYBD(Adresse,Objekt1,Objekt2,Char$,<<N_objc,<<N_char$)

- *Von FORM_DO verwendete Unterroutine!* -

Behandelt Texteingaben über Tastatur bei editierbaren Text-Objekten (siehe OBJC_EDIT). Zur eigenen Verwendung ist diese Funktion nur bedingt nutzbar, da eine eigene Formular-Verwaltung nach Aufruf von OBJC_DRAW unverhältnismäßig aufwendig ist (umfangreiche Abfrageschleife).

Back	0 = Exit-Objekt (EXIT bzw. TOUCHEXIT) wurde gewählt. > 0 = Formular-Dialog ist noch nicht beendet.
Adresse	Adresse des bearbeiteten Objektbaumes (Formulars).
Objekt1	Nummer des aktuell editierten Text-Objektes.
Objekt2	Nummer des nächsten editierbaren Text-Objektes.
Char$	ASCII-Zeichen, das von der Tastatur gelesen wurde.
N_objc	Rückgabevariable, die nach Abschluß die Nummer des für den nächsten Aufruf aktualisierten Textobjektes enthält.
N_char$	String-Rückgabevariable, die nach Abschluß das nächste Zeichen des Text-Objektes enthält. Wurde eine Sondertaste (z.B. Cursor-Taste) gewählt, enthält N_char$ den Wert 0.

18.6 Fileselect-Manager

FSEL_INPUT() — Fileselect-Box produzieren

Back=FSEL_INPUT(<<Pfad$>>,<<Datei$>>,<<Button)

Erstellt eine Fileselect-Box (vgl. FILESELECT) und übernimmt die Benutzerkontrolle während der Dateiauswahl. Aus Gründen der Bedienungssicherheit ist FILESELECT vorzuziehen, da bei (versehentlich) unkorrekten String-(Puffer-)angaben die GEM-Verwaltung von File-

select-Boxen unwiderruflich ins Schleudern geraten kann und eine fehlerfreie Dateiauswahl mit diesem Formular fraglich wird.

Von der Funktion wird automatisch FORM_DIAL(0) und FORM_DIAL(3) ausgeführt. Betreffend der Hintergrund-Restauration gelten hier dieselben Ausführungen wie zu FORM_DIAL.

Back	**0 = Fehler; <> 0 = Funktion korrekt ausgeführt.**
Pfad$	**Vorbereitete String-Rückgabevariable. Enthält bei Funktionsaufruf die komplette Bezeichnung des gewünschten Zugriffspfads (minimal "*.*"). Nach Funktionsende ist hier der vom Benutzer tatsächlich gewählte Pfadname eingetragen.**
Datei$	**Vorbereitete String-Rückgabevariable. Enthält bei Funktionsaufruf ggfs. den Namen einer vorgegebenen Datei, der im Formular unter "Auswahl:" in die Eingabezeile eingetragen wird (minimal ""). Nach Funktionsende ist hier der vom Benutzer tatsächlich gewählte Dateiname eingetragen.**
Button	**Rückgabe-Variable (vom Benutzer gewähltes Exit-Objekt): 0 = ABBRUCH wurde angeklickt. 1 = OK wurde angeklickt, bzw. <Return> wurde gedrückt, bzw. ein Dateiname im Auswahlfenster wurde durch Doppelklick gewählt.**

18.7 Graphics-Manager

GRAF_DRAGBOX() — Schiebebox produzieren

Back=GRAF_DRAGBOX(Litb,Lith,Xsta,Ysta,Xpos,Ypos,Breite,Höhe,...
...<<Lastx,<<Lasty)

Erzeugt eine Schiebebox mit vorgegebener Größe innerhalb eines bestimmten Begrenzungsrechtecks. Die kleinere Box folgt bei gedrückt gehaltener linker Maustaste dem Mauszeiger. Sie läßt sich dabei jedoch nicht über die vorgegebenen Grenzen des umgebenden Rahmens - der von der Funktion nicht gezeichnet wird - hinausbewegen.

Wird die linke Maustaste losgelassen, gilt die Funktion damit als beendet. Daraus folgt, daß die Funktion nur bei gedrückt gehaltener linker Maustaste aufgerufen werden kann. Als Rückgabeparameter erhält man nach Abschluß die absolute X- und Y-Koordinate der linken oberen Ecke der kleinen Box zum Zeitpunkt des Loslassens der Maustaste.

Back	**0 = Fehler; <> 0 = Funktion korrekt ausgeführt.**
Litb	**Gewünschte Breite der beweglichen Schiebebox.**
Lith	**Gewünschte Höhe der beweglichen Schiebebox.**
Xsta	**X-Startkoordinate der linken oberen Ecke der beweglichen Box.**

Ysta	Y-Startkoordinate der linken oberen Ecke der beweglichen Box.
Xpos	X-Koordinate der linken oberen Ecke der Rahmenbox.
Ypos	Y-Koordinate der linken oberen Ecke der Rahmenbox.
Breite	Unveränderliche Breite der Rahmenbox.
Höhe	Unveränderliche Höhe der Rahmenbox.
Lastx	Rückgabevariable (Schiebebox-X-Koordinate bei Funktionsende).
Lasty	Rückgabevariable (Schiebebox-Y-Koordinate bei Funktionsende).

Beispiel:

```
DO
  REPEAT
  UNTIL MOUSEK
  SETMOUSE 50,50
  ~GRAF_DRAGBOX(50,30,50,50,40,40,200,200,xb%,yb%)
  BOX xb%,yb%,xb%+50,yb%+30
LOOP
```

GRAF_GROWBOX() Box-Vergrößerungseffekt produzieren

Back=GRAF_GROWBOX(Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)

- Von FORM_DIAL verwendete Unterroutine! -

Erzeugt einen Bewegungseffekt. Eine Box mit vorgegebener Startgröße bewegt sich dabei von einer Bildschirmposition zu einer anderen Bildschirmposition mit einer vorgegebenen größeren Endgröße. Der Bildschirminhalt wird dadurch nicht beeinflußt.

Back 0 = Fehler; <> 0 = Funktion korrekt ausgeführt.

Litx, Lity, Litb, Lith:

Koordinaten und Maße des kleineren (Start-)Rechtecks. Werden die Parameter Litb und Lith mit 0 belegt, wird der Eindruck erzeugt, daß das Zentrum des größeren (Ziel-)Rechtecks die Startkoordinate des kleineren (Start-)Rechtecks ist.

Bigx, Bigy, Bigb, Bigh:

Koordinaten und Maße des größeren (Ziel-)Rechtecks.

Beispiel:

```
FOR i%=1 TO 25
  DEFFILL ,2,i%
  BOUNDARY 1
  PBOX i%*10,10,i%*10+20,30
  ~GRAF_GROWBOX(i%*10,10,20,20,270-i%*10,150,50,50)
  '  ~GRAF_SHRINKBOX(270-i%*10,150,50,50,i%*10,10,20,20)
  PBOX 270-i%*10,150,320-i%*10,200
  BOUNDARY 0
  DEFFILL ,0,0
  PBOX i%*10,10,i%*10+20,30
NEXT i%
```

GRAF_HANDLE() VDI-Handle u. Zeichenmaße ermitteln

Back=GRAF_HANDLE(<<Ch_breite,<<Ch_höhe,<<Bx_breite, <<Bx_höhe)

Ermittelt das VDI-Handle der aktuellen VDI-Workstation und die Zeichen- und Zeichenbox-Maße des aktuellen Standard-Fonts (Hires = 8x16).

Back	Aktuelles VDI-Handle.
Ch_breite	Rückgabevariable (aktuelle Standard-Zeichenbreite).
Ch_höhe	Rückgabevariable (aktuelle Standard-Zeichenhöhe).
Bx_breite	Rückgabevariable (aktuelle Zeichenboxbreite).
Bx_höhe	Rückgabevariable (aktuelle Zeichenboxhöhe).

GRAF_MKSTATE() Umschalttasten- und Maus-Status ermitteln

Back=GRAF_MKSTATE(<<Mausx,<<Mausy,<<Mausk,<<Switch)

Ermittelt die aktuellen Mauskoordinaten, den Maustastenstatus (vgl. MOUSE), sowie den Status der Umschalttasten (<Shift>/<Control>/<Alternate><CapsLock> siehe auch BIOS(11)).

Back	Ohne Bedeutung (immer 1).
<<	Die Rückgabe-Variablen Mausx, Mausy, Mausk und Switch haben dieselbe Bedeutung wie bei EVNT_BUTTON.

GRAF_MOUSE() Mausform bestimmen (DEFMOUSE)

Back=GRAF_MOUSE(Maus,Adresse)

Diese Funktion kann komplett durch die Befehle DEFMOUSE und HIDEM/SHOWM ersetzt werden. Sie ermöglicht die Auswahl einer Mausform bzw. die Benutzerdefinition einer Mausform. Außerdem kann der Mauszeiger hiermit an- und ausgeschaltet werden.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Maus	Mausformindex: 0 - 7 = Siehe DEFMOUSE Form. 255 = Flag zeigt an, daß der ab Adresse liegende Bit-Block mit den Format- und Musterdaten einer selbstdefinierten Mausform installiert werden soll. 256 = Mauszeiger aus 257 = Mauszeiger an
Adresse	Startadresse eines 37 Words großen Speicherbereichs, in dem die Format- und Musterdaten der selbstdefinierten Mausform abgelegt sind (Aufbau siehe DEFMOUSE Maus$).

GRAF_MOVEBOX() Box-Bewegungseffekt produzieren

Back=GRAF_MOVEBOX(Breite,Höhe,Xsta,Ysta,Xziel,Yziel)
Erzeugt einen Bewegungseffekt. Eine Box mit gleichbleibender Größe bewegt sich dabei von einer Bildschirmposition zu einer anderen Bildschirmposition. Der Bildschirminhalt wird dadurch nicht beeinflußt.

Back	**0 = Fehler; <> 0 = Funktion korrekt ausgeführt.**
Breite	**Unveränderliche Breite der Box.**
Höhe	**Unveränderliche Höhe der Box.**
Xsta	**X-Startposition der linken oberen Ecke der Box.**
Ysta	**Y-Startposition der linken oberen Ecke der Box.**
Xziel	**X -Zielposition der linken oberen Ecke der Box.**
Yziel	**Y -Zielposition der linken oberen Ecke der Box.**

GRAF_RUBBERBOX() Gummiband-Box (Lasso) produzieren

Back=GRAF_RUBBERBOX(Xpos,Ypos,Bmin,Hmin,<<Lastb,<<Lasth)
Erzeugt einen "Lasso"-Effekt. D.h. es wird die Position der linken oberen Ecke eines Rechtecks, sowie die kleinstmögliche Höhe und Breite des Lassos angegeben. Die rechte untere Ecke des Lassos folgt bei gedrückt gehaltener linker Maustaste den Mausbewegungen, während die linke obere Ecke an der definierten Position stehenbleibt. Wird die linke Maustaste losgelassen, gilt die Funktion damit als beendet. Daraus folgt, daß die Funktion nur bei gedrückt gehaltener linker Maustaste aufgerufen werden kann. Als Rückgabeparameter erhält man nach Abschluß die Breite und Höhe des Rechtecks zum Zeitpunkt des Loslassens der linken Maustaste.

Back	**0 = Fehler; <> 0 = Funktion korrekt ausgeführt.**
Xpos	**X-Koordinate der linken oberen Ecke des Lassos.**
Ypos	**Y-Koordinate der linken oberen Ecke des Lassos.**
Bmin	**Kleinstmögliche Lasso-Breite in Pixel.**
Hmin	**Kleinstmögliche Lasso-Höhe in Pixel.**
Lastb	**Rückgabevariable (Lasso-Breite bei Funktionsende).**
Lasth	**Rückgabevariable (Lasso-Höhe bei Funktionsende).**

GRAF_SHRINKBOX() Box-Verkleinerungseffekt produzieren

Back=GRAF_SHRINKBOX(Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)
- Von FORM_DIAL verwendete Unterroutine! -
Erzeugt einen Bewegungseffekt. Eine Box mit vorgegebener Startgröße bewegt sich dabei von einer Bildschirmposition zu einer anderen

Bildschirmposition mit einer vorgegebenen kleineren Endgröße. Der Bildschirminhalt wird dadurch nicht beeinflußt.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Litx, Lity, Litb, Lith:	Koordinaten und Maße des kleineren (Ziel-)Rechtecks. Werden die Parameter Litb und Lith mit 0 belegt, wird derEindruck erzeugt, daß das Zentrum des größeren (Start-)Rechtecks die Zielkoordinate des kleineren (Ziel-)Rechtecks ist.
Bigx, Bigy, Bigb, Bigh:	Koordinaten und Maße des größeren (Start-)Rechtecks.

Siehe Beispiel zu GRAF_GROWBOX().

GRAF_SLIDEBOX() Schiebebox innerhalb von Formularen

Back=GRAF_SLIDEBOX(Adresse,Rahmen,Slider,Flag)

- Nur innerhalb von Formularen verwendbar! -

Der Umriß eines Formular-Objekts (Slide-Box) wird innerhalb eines Rahmen-Objektes bei gedrückt gehaltener linker Maustaste horizontal oder vertikal entsprechend den Mausbewegungen verschoben. Diese Routine wird z.B. vom GEM dazu verwendet, um Schiebebox-Effekte bei den H_slide- und V_slide-Objekten im rechten und unteren Randbereich eines Fensters zu zu erzeugen.

Wird die linke Maustaste losgelassen, gilt die Funktion damit als beendet. Daraus folgt, daß die Funktion nur bei gedrückt gehaltener linker Maustaste aufgerufen werden kann.

Durch die Funktion wird nicht in die Objektbaumstruktur eingegriffen, es wird lediglich der Schiebeeffekt produziert. Das Neuzeichnen des bewegten Objektes an der letzten Slider-Position (Koordinatenanpassung und anschließendes OBJC_DRAW) bleibt dem Programmierer überlassen. Das Rahmen-Objekt, innerhalb dessen sich das Schiebe-Objekt bewegt, muß im betreffenden Objektbaum ein Eltern-Objekt des Schiebers sein.

Back	Stellung des Schiebermittelpunktes bei Loslassen der linken Maustaste (Funktionsende), relativ zum umgebenden Rahmen. 0 = Slider ist ganz oben bzw. ganz links. bis 1000 = Slider ist ganz unten bzw. ganz rechts.
Adresse	Adresse des Objektbaumes, dem die beiden Objekte angehören.
Rahmen	Nummer des Rahmen-Objektes.
Slider	Nummer des Schieber-Objektes.

Flag	Schieberichtung: 0 = Horizontal (H_slide); 1 = Vertikal (V_slide)

GRAF_WATCHBOX() Objektstatus gemäß Mausposition

(in/out)

Back=GRAF_WATCHBOX(Adresse,Objekt,Stat_i,Stat_o)

- Nur innerhalb von Formularen verwendbar! -

Ein Formular-Objekt wird daraufhin überwacht, ob sich der Mauszeiger bei gedrückt gehaltener linker Maustaste innerhalb oder ausserhalb dieses Objektes befindet.

Wird die linke Maustaste losgelassen, gilt die Funktion damit als beendet. Daraus folgt, daß die Funktion nur bei gedrückt gehaltener linker Maustaste aufgerufen werden kann.

Back	Flag, das anzeigt, ob sich bei Loslassen der linken Maustaste (Funktionsende) der Mauszeiger innerhalb (1) oder ausserhalb (0) des zu überwachenden Objekts befunden hat.
Adresse	Adresse des Objektbaumes, dem das zu überwachende Objekt angehört.
Objekt	Nummer des zu überwachenden Objektes.
Stat_i	Status, den das Objekt annehmen soll, sobald der Mauszeiger bei gedrückter Maustaste das Objekt betritt (siehe OB_STATE).
Stat_o	Status, den das Objekt annehmen soll, sobald der Mauszeiger bei gedrückter Maustaste das Objekt verläßt (siehe OB_STATE).

18.8 Menü-Manager

MENU_BAR() Zeichnen/Löschen einer Menüleiste

Back=MENU_BAR(Adresse,Modus)

Zeichnet oder löscht eine Menüleiste, die als Typ "Menü" in einem Objektbaum gespeichert ist.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Menübaumes.
Modus	Funktionsmodus: 0 = Menüleiste löschen (vgl. MENU KILL) 1 = Menüleiste zeichnen (vgl. MENU menütext$()).

MENU_ICHECK() Menü-Checkmark zeichnen/löschen

Back=MENU_ICHECK(Adresse,Objekt,Modus)
Setzt oder löscht ein Checkmark (Häkchen) vor einem Menüpunkt (vgl. MENU menü). Das Menü muß als Typ "Menü" in einem Objektbaum gespeichert sein. Ggfs. ist bei der Menütext-Definition vor dem entsprechenden Menüpunkt Platz (zwei Leerzeichen) für das Checkmark vorzusehen.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Menübaumes.
Objekt	Objektnummer der Zeile innerhalb des Baumes.
Modus	Funktionsmodus: 0 = Checkmark löschen (vgl. MENU Menü,0). 1 = Checkmark setzen (vgl. MENU Menü,1).

MENU_IENABLE() Menüeinträge aktivieren/deaktivieren

Back=MENU_IENABLE(Adresse,Objekt,Modus)
Aktiviert (schwarze Schrift = wählbar) oder deaktiviert (graue Schrift = nicht wählbar) eine Menüzeile (vgl. MENU menü). Das Menü muß als Typ "Menü" in einem Objektbaum gespeichert sein.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Menübaumes.
Objekt	Objektnummer der Zeile innerhalb des Baumes.
Modus	Funktionsmodus: 0 = Zeile deaktivieren (vgl. MENU Menü,2). 1 = Zeile aktivieren (vgl. MENU Menü,3).

MENU_TEXT() Menü-Text anpassen

Back=MENU_TEXT(Adresse,Objekt,Text$)
Übergabe einer Menütextzeile als Ersatz einer bereits bestehenden Zeile. Das Menü muß als Typ "Menü" in einem Objektbaum gespeichert sein.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Menübaumes.
Objekt	Nummer d. zu ändernden Zeile innerhalb des Baumes.
Text$	Neuer Menüzeilentext. Die Länge des neuen Textes darf die Länge des alten Textes nicht überschreiten (Absturzgefahr).

MENU_TNORMAL() — Menü-Titel invers/normal

Back=MENU_TNORMAL(Adresse,Objekt,Modus)

Ermöglicht inverse oder normale (schwarz auf weiß) Darstellung einer Menütitelzeile.

Das Menü muß als Typ "Menü" in einem Objektbaum gespeichert sein.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Menübaumes.
Objekt	Objektnummer der Titelzeile innerhalb des Baumes.
Modus	Funktionsmodus: 0 = Zeile invers darstellen. 1 = Zeile normal darstellen (vgl. MENU OFF).

MENU_REGISTER() — Accessory in Register eintragen

Back=MENU_REGISTER(Handle,Text$)

Meldet ein Accessory im Accessory-Register an und trägt den Menütitel im Desktop-Menü ein.

Back	Index (0 - 5) des Nameneintrags im Desktop-Menü. Es sind maximal 6 Accessories gleichzeitig einsetzbar. Ist kein Menüplatz mehr verfügbar, erhält man eine -1.
Handle	AES-Handle (ap_id siehe APPL_INIT) des Accessories.
Text$	Menüzeilentext des Eintrags im Desktop-Menü.

18.9 Object-Manager

OBJC_ADD() — Objekt in Baum einfügen

Back=OBJC_ADD(Adresse,Elter,Kind)

Integriert ein Objekt komplett in einen bestehenden Objektbaum. Dabei werden in GFA-BASIC im Gegensatz zur echten GEMSYS-Funktion außer OB_NEXT auch die zu erweiternden Strukturzeiger OB_HEAD und OB_TAIL angepaßt.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des Objektbaumes, in den das Objekt einzuordnen ist.
Elter	Nummer des übergeordneten (Eltern-)Objektes, dem das neue (Kind-) Objekt untergeordnet wird.
Kind	Nummer des neu zuzuordnenden (Kind-)Objektes, das dem bestehenden (Eltern-)Objekt unterzuordnen ist.

OBJC_CHANGE() — Objekt-Status ändern

Back=OBJC_CHANGE(Adresse,Objekt,Dummy,Xpos,Ypos,Breite,Höhe,..

...Status,Modus)

- Von FORM_DO verwendete Unterroutine! -
Ermöglicht die Änderung des Status (OB_STATE) eines Objekts und ggfs. dessen Neuzeichnung innerhalb eines beliebigen Begrenzungsrechtecks.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des Objektbaumes, dem das zu ändernde Objekt angehört.
Objekt	Nummer des Objektes, dessen OB_STATE geändert werden soll.
Dummy	Ohne Bedeutung (immer 0).
Xpos, Ypos, Breite, Höhe:	Koordinaten und Maße eines Bildschirmausschnitts, über dessen Grenzen hinaus nicht gezeichnet werden soll. So kann die Darstellung des Objekts z.B. auf einen bestimmten Window-Bereich beschränkt werden (vgl. CLIP).
Status	Gewünschter neuer Objektstatus (siehe OB_STATE).
Modus	Funktionsmodus: 0 = Geändertes Objekt nicht neu zeichnen 1 = Geändertes Objekt neu zeichnen

OBJC_DELETE() — Objekt deaktivieren

Back=OBJC_DELETE(Adresse,Objekt)

Löscht ein Objekt aus einem bestehenden Objektbaum. D.h., es werden OB_HEAD, OB_TAIL und OB_NEXT des Objektes so gesetzt, daß es nicht mehr wählbar ist. Werden die Zeiger wieder restauriert, bzw. das Objekt durch OBJC_ADD neu integriert, kann es wieder angewählt werden.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des Objektbaumes, dem das zu löschende Objekt zugeordnet ist.
Objekt	Nummer des zu löschenden Ziel-Objekts.

OBJC_DRAW() — Objekt(e) darstellen

Back=OBJC_DRAW(Adresse,Ebene1,Kinder,Xpos,Ypos,Breite,Höhe)

Stellt grafisch einen kompletten Objektbaum (Formular) oder einzelne Ebenen innerhalb bestimmter Koordinaten auf dem Bildschirm dar.

Einzelne Objekte aus einer Ebene können nicht einzeln gezeichnet werden. Es kann jedoch eine Startebene angegeben werden, ab welcher die zu zeichnenden weiteren Ebenen gezählt werden.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des gewünschten Objektbaumes.
Ebene1	Nummer der ersten darzustellenden Baum-Ebene.
Kinder	Anzahl der weiteren Unter-Ebenen, die gezeichnet werden sollen: 0 = Nur Ebene1 zeichnen 1 = Ebene1 + Kind-Ebene 2 = Ebene1 + Kind-Ebene + Kind-Kind-Ebene etc.

Xpos, Ypos, Breite, Höhe:
Koordinaten und Maße eines Bildschirmausschnitts, über dessen Grenzen hinaus nicht gezeichnet werden soll. So kann die Darstellung von Objekten z.B. auf einen bestimmten Window-Bereich beschränkt werden (vgl. CLIP).

OBJC_EDIT() — Objekt-Texteingabe

Back=OBJC_EDIT(Adresse,Objekt,Char$,Zpos,Modus,<<N_pos)

Bei Objekten des Typs G_text und G_boxtext kann hiermit die Eingabe und Edition des Textes innerhalb des Objektes durchgeführt werden. Die Funktion ist ggfs. mehrmals hintereinander mit entsprechend geänderten Parametern auszuführen.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des Objektbaumes, dem das Textobjekt angehört.
Objekt	Nummer des Textobjektes, das bearbeitet werden soll.
Char$	ASCII-Zeichen, das in den Objekttext eingefügt werden soll.
Zpos	Zielposition des neuen Zeichens im Objekttext.
Modus	Funktionsmodus: 1 = Formatierten String berechnen (Vorgabetext mit Textmaske verknüpfen) und Cursor einschalten (Objekt-Init). 2 = Neues Zeichen Char$ einlesen, in Objekttext einfügen und Objekttext neu ausgeben. 3 = Cursor ausschalten, Eingabe/Edition beenden.
N_pos	Rückgabe-Variable, die nach Abschluß die nächste Zeichenposition innerhalb des Objekttextes enthält.

OBJC_FIND() — Objektnummer ermitteln

Back=OBJC_FIND(Adresse,Objekt1,Ebenen,Xpos,Ypos)

- Von FORM_DO verwendete Unterroutine! -

Ermittelt die Objektnummer eines Objektes an einer bestimmten Bildschirmposition. Dies ist z.B. nötig, wenn durch Mausklick ein Objekt in einem selbstverwalteten Formular gewählt wurde.

Back	Nummer des ermittelten Objektes oder -1, wenn sich an der Position kein Objekt befindet.
Adresse	Adresse des Objektbaumes, dem das zu Objekt zugeordnet ist.
Objekt1	Nummer des Objektes, mit dem die Suche beginnen soll.
Ebenen	Anzahl der Ebenen, die ab Objekt1 zu untersuchen sind: 0 = Nur Eltern-Ebene von Objekt1 untersuchen. 1 = Eltern-Ebene von Objekt1 + nächste Ebene. etc.
Xpos	X-Koordinate des zu ermittelnden Objekts.
Ypos	Y-Koordinate des zu ermittelnden Objekts.

OBJC_OFFSET() — Absolute Objektposition ermitteln

Back=OBJC_OFFSET(Adresse,Objekt,<<Xpos,<<Ypos)

Berechnet die Koordinaten der linken oberen Ecke eines bestimmten Objektes, bezogen auf die linke obere Ecke des Bildschirms.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Adresse	Adresse des Objektbaumes, dem das Zielobjekt zugeordnet ist.
Objekt	Nummer des Objektes, dessen Position berechnet werden soll.
Xpos	Rückgabevariable, die nach Abschluß die berechnete X-Koordinate des betreffenden Objekts enthält.
Ypos	Rückgabevariable, die nach Abschluß die berechnete Y-Koordinate des betreffenden Objekts enthält.

OBJC_ORDER() — Objekt neu zuordnen

Back=OBJC_ORDER(Adresse,Objekt,Ebene)

Verschiebt ein Objekt innerhalb eines Objektbaumes auf eine andere Objekt-Ebene und paßt die entsprechenden Zeiger an.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt
Adresse	Adresse des Objektbaumes, dem das zu verschiebende Objekt angehört.
Objekt	Nummer des Objektes, das verschoben werden soll.
Ebene	Nummer der Ebene, welcher das Objekt neu zugeordnet werden soll. -1 = Eine Ebene aufwärts 0 = Unterste Ebene 1 = Unterste Ebene +1 2 = Unterste Ebene +2 etc.

18.10 Resource-Manager

RSRC_FREE() — Resource-Speicher freigeben

Back=RSRC_FREE()

Der durch RSRC_LOAD von GEM reservierte (MALLOC-) Speicher wird wieder freigegeben (MFREE) und der entsprechende Objektbaum gelöscht. Ggfs. ist anschließend der BASIC-Speicher zu restaurieren (siehe RSRC_LOAD).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.

RSRC_GADDR() — Resource-Adresse ermitteln

Back=RSRC_GADDR(Typ,Objekt,<<Adresse)

Ermittelt die Startadresse eines durch RSRC_LOAD geladenen Objektbaumes, bzw. die Adresse eines darin enthaltenen FORM_ALERT-Strings. Gemäß Digital-Research-Dokumentation sollten noch weitere Strukturadressen hierdurch erfahrbar sein. Nach Angabe verschiedener Veröffentlichungen zu dieser Funktion und eigener Erfahrung muß erwähnt werden, daß dies nicht der Fall ist. Nur die Modi 0 und 15 arbeiten fehlerfrei.

Um die übrigen Strukturzeiger zu ermitteln, bedienen Sie sich bitte der Objekt-Variable OB_ADR(). Mit Kenntnis der Objekt- und Unterstrukturen lassen sich damit alle interessanten Adressen in Erfahrung bringen.

Back — 0 = Fehler; <> 0 = Funktion korrekt ausgeführt.

Typ — Strukturtyp, dessen Adresse gesucht wird:

```
 0 = TREE       (Objektbaum         -> Fehlerfrei)
15 = AD_FRSTR   (FORM_ALERT-String -> Fehlerfrei)
 1 = OBJECT      2 = TEDINFO     3 = ICONBLK    ---.
 4 = BITBLK      5 = STRING      6 = IMAGEDATA     ! Arbeiten
 7 = OBSPEC      8 = TE_PTEXT    9 = TE_PTMPLT     !- nicht
10 = TE_PVALID  11 = IB_PMASK   12 = IB_PDATA      ! korrekt!
13 = IB_PTEXT   14 = BI_PDATA   16 = AD_FRIMG   ---'
```

Objekt — Nummer des Objektes, dessen Strukturadressen ermittelt werden sollen.

Adresse — Rückgabe-Variable, die nach Abschluß die Adresse der gewünschten Struktur enthält.

RSRC_LOAD() Resource-Datei laden

Back=RSRC_LOAD(Rsc_name$)

Es wird ein Objektbaum in Form einer GEM-Resource-Datei (.RSC-File) in den Speicher geladen und die Zeichen-Koordinaten (8x16- bzw. 8x8-Font) werden in Pixel-Koordinaten umgerechnet. Das GEM reserviert als Speicher für .RSC-Files den dafür benötigten Teil des momentan verfügbaren Restspeichers (in GFA-BASIC: MALLOC oberhalb von HIMEM). Eine unzureichende Größe dieses Restspeichers führt dazu, daß die Datei nicht geladen wird.

Deshalb ist vor RSRC_LOAD genügend Speicherplatz zu reservieren:

```
RESERVE FRE(0)-33000
```

Der maximale Speicherbedarf einer .RSC-Datei liegt bei 33000 Byte.

Die Funktion lädt zuerst von der evtl. im Pfad vorgegebenen Station und sucht - wenn ohne Erfolg - ggfs. noch auf Station A. Bei programminterner Resource-Definition (z.B. in Datas) müssen ggfs. die zeichenorientierten Koordinaten durch RSRC_OBFIX auf Pixel-Format umgerechnet werden.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Rsc_name$	Dateiname des gewünschten Resource-Files.

RSRC_OBFIX() Objektkoordinaten umwandeln

Back=RSRC_OBFIX(Adresse,Objekt)

Wurde ein Objektbaum mit zeichenorientierten Koordinaten direkt im Speicher erzeugt, müssen vor Aufruf eines Objektes die zeichenorientierten Koordinaten OB_X, OB_Y, OB_W und OB_H (8x16- bzw. 8x8-Font) in das Pixel-Format umgerechnet werden. Diese Umrechnung, sowie die Eintragung der geänderten Werte in die Objektstruktur wird von RSRC_OBFIX durchgeführt. Bei geladenen Objektbäumen werden die in .RSC-Files grundsätzlich verwendeten Zeichenkoordinaten durch RSRC_LOAD für alle Objekte während des Ladens automatisch angepaßt.

Back	Ohne Bedeutung (immer 1).
Adresse	Adresse des Objektbaumes, dem das betreffende Objekt angehört.
Objekt	Nummer des Objektes, dessen Koordinaten umgerechnet werden sollen.

RSRC_SADDR() Resource-Adresse einfügen

Back=RSRC_SADDR(Typ,Objekt,Zeiger)

Ändern eines Strukturzeigers innerhalb einer Objektstruktur. Werden Strukturdaten oder -blöcke (z.B. TEDINFO) im Speicher verschoben, kann hiermit der entsprechende Zeiger innerhalb der betreffenden Objektstruktur angepaßt werden.

Auch diese Funktion arbeitet nicht immer fehlerfrei, so daß ggfs. die Zeiger durch LPOKEs in die Struktur geschrieben oder durch die Objekt-Variablen (OB_ADR, OB_NEXT, OB_SPEC etc.) bestimmt werden müssen.

Back	0 = Fehler ; <> 0 = Funktion korrekt ausgeführt.
Typ	Strukturtyp, dessen Adresse gesetzt werden soll: 0 = TREE 1 = OBJECT, 2 = TEDINFO, 3 = ICONBLK, 4 = BITBLK 5 = STRING, 6 = IMAGEDATA, 7 = OBSPEC, 8 = TE_PTEXT 9 = TE_PTMPLT, 10 = TE_PVALID, 11 = IB_PMASK, 12 = IB_PDATA 13 = IB_PTEXT, 14 = BI_PDATA, 16 = AD_FRIMG, 15 = AD_FRSTR
Objekt	Nummer des Objektes, in dessen Struktur der betr. Zeiger liegt.
Zeiger	Neuer Zeiger auf die geänderte Adresse.

18.11 Scrap-Manager

SCRP_READ() Globalen GEM-Puffer (Clipboard) lesen

Back=SCRP_READ(<<Puffer$>>)

Das GEM unterhält für den interaktiven Datenaustausch zwischen verschiedenen Programmen (Applikationen) einen ca. 160 Byte großen globalen Puffer, der von der Speicherverwaltung bei Programmstart und -ende nicht beinflußt wird (ab $9226 -> TOS/6.2.86). Dieser Puffer kann für die Lagerung/Übergabe beliebiger Daten verwendet werden (Clipboard/schwarzes Brett).

Mit SCRP_READ kann nun der Inhalt dieses Clipboards in einen Text-Puffer kopiert/gelesen werden. Ein Zweck eines solchen Clipboards besteht zum Beispiel darin, Texte/Dateipfade/Variableninhalte an ein durch CHAIN nachgeladenes Programm übergeben zu können, das sich die gelagerten Daten bei Bedarf aus diesem Puffer wieder auslesen kann. Darüber hinaus steht die Nutzung dieses Puffers dem Programmierer frei.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Puffer$	Vorbereitete String-Variable (Puffer$=SPACE$(160)), die nach Abschluß den gelesenen String enthält. Als Puffer-Endkennung wird dem String durch SCRP_WRITE ein Null-Byte angehängt.

```
Inhalt$=LEFT$(Puffer$,INSTR(Puffer$,CHR$(0))-1)
```

SCRP_WRITE() Globalen GEM-Puffer (Clipboard) schreiben

Back=SCRP_WRITE(Text$)

Es wird ein beliebiger Text-String (max. 160 Bytes) in den globalen Scrap-Puffer geschrieben (siehe Erläuterungen zu SCRP_READ).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Text$	Beliebiger String (max. Größe = 160 Byte).

18.12 Shell-Manager

SHEL_ENVRN() DOS-Environment-Adresse ermitteln

Back=SHEL_ENVRN(<<Adresse,String$)

Diese Funktion liefert die Adresse des DOS-Environment-Strings (voreingestellter Suchpfad: PATH=;A:\ ), bzw. eines Teils daraus (vgl. BASEPAGE+44).

Back	Offset, der (um 1 inkrementiert) von der gelieferten Adresse abgezogen werden muß. Ab Adresse-(Back+1) ist der angegebene Such-String im Speicher zu finden. Ist der Such-String nicht im DOS-Environment-String enthalten, enthält Back eine Null und Adresse den ASCII-Wert des ersten Zeichens des Such-Strings.
Adresse	Rückgabevariable, die nach Abschluß Environment-Adresse enthält.
String$	Anzugebender Such-String (z.B. "A:\", "PATH", "=;").

Zur Verdeutlichung:

```
Offs%=SHEL_ENVRN(Adrs%,"PATH=;A:\")
Buff$=SPACE$(20)
Strt%=Adrs%-(Offs%+1)
BMOVE Strt%,VARPTR(Buff$),20
Envrn$=LEFT$(Buff$,INSTR(Buff$,CHR$(0))-1)
PRINT "Ab ";Strt%;" -> ";Envrn$
```

SHEL_FIND() — Applikations-Namen auf Disk suchen

Back=SHEL_FIND(< <Pfad$> >)

Sucht auf der angegebenen bzw. aktuellen Station im angegebenen bzw. aktuellen Ordner nach dem angegebenen File. Wird die Datei auf der betreffenden Station nicht gefunden, sucht die Funktion auch noch auf Disk A. Hier wird dann nur noch im Haupt-Direktory gesucht.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Pfad$	Vorzubereitende Rückgabe-String-Variable, die vor Aufruf den Namen (ggfs. incl Pfad) des gesuchten Files enthält. Nach Abschluß wird von der Funktion im Erfolgsfall (Back > 0) hier die komplette Pfadbezeichnung der gefundenen Datei abgelegt. Verwendete Wildcards (*?) werden von der Funktion nicht ersetzt. Sollten sich bei der Pfadrückgabe größere Längen ergeben als bei der Übergabe, kann es zu Fehlfunktionen kommen. Daher sollte die String-Variable durch SPACE$ ausreichend erweitert werden:

```
Übergabe :   ":\Ordner\Datei.*"+SPACE$(20)
Rückgabe :   Back  -> 1
             Pfad$ -> "A:\Ordner\Datei.*"
                      (+Null-Byte als Abschluß)
```

SHEL_GET() — Environment-Puffer lesen

Back=SHEL_GET(Anzahl,< <Puffer$)

Liest eine vorgegebene Byte-Anzahl aus dem aktuellen Environment-Puffer. In diesem Environment-Puffer ist das jeweils aktuelle Desktop.Inf-File gespeichert, das Angaben über verschiedene Betriebszustände, sowie über den Aufbau des Desktops enthält.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Anzahl	Anzahl der zu lesenden Bytes (max. 518 = 19 Zeilen).
Puffer$	Rückgabe-String-Variable, die nach Abschluß gelesene Bytes enthält.

SHEL_PUT() — In Environment-Puffer schreiben

Back=SHEL_PUT(Anzahl,Envrn$)

Schreibt eine vorgegebene Byte-Anzahl in den aktuellen Environment-Puffer (siehe SHEL_GET).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Anzahl	Anzahl der zu schreibenden Bytes (max. 518 = 19 Zeilen).
Envrn$	String-Variable, die die zu schreibenden Bytes enthält.

SHEL_READ() Applikationsnamen und -kommandozeile lesen

Back=SHEL_READ(<<Name$,<<Comm$)

Übergibt den File-Namen (ohne Pfad), mit dem die aktuelle Applikation von einer übergeordneten Ebene (Desktop, EXEC, CHAIN) aufgerufen wurde und die dabei ggfs. übergebene Kommandozeile (max. 128 Byte).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Name$	Rückgabe-String-Variable, die nach Abschluß den Programmnamen enthält (Null-Byte als Endmarkierung).
Comm$	Rückgabe-String-Variable, die nach Abschluß Kommandozeile enthält (Null-Byte als Endmarkierung). Mögliche Kommandos: - Bei .TTP-Programmen der bei Start eingegebene Text. - Name des bei "Anwendung anmelden" eingetragenen Files. - Inhalt von BASEPAGE+128 bei Compiler-CHAIN. - Bei EXEC angegebene beliebige Kommandozeile. - etc.

SHEL_WRITE() Applikation anmelden/starten

Back=SHEL_WRITE(Modus,Grafik,Gemflag,Comm$,Name$)

Hiermit kann ein beliebiges Programm gestartet werden. Anders als bei EXEC wird jedoch der Speicherplatz des aufrufenden Programms vorher freigegeben. Es bleibt also nicht resident. Die Compiler-Version des Befehls CHAIN verwendet z.B. diese Funktion, um ein Programm nachzuladen.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Modus	Funktionsmodus: 0 = GEM verlassen (Rückkehr zum Desktop?) 1 = Programm laden und starten
Grafik	Neues Programm arbeitet: 0 = Ohne Grafik 1 = Mit Grafik
Gemflag	Neues Programm ist: 0 = .TOS/.TTP-Programm 1 = GEM-Applikation (.PRG)
Comm$	String-Variable, die die zu übergebende Kommandozeile enthält (minimal CHR$(0)). Das Compiler-CHAIN legt diese String-Adresse auf BASEPAGE+128.
Name$	String-Variable, die den Namen des zu startenden Programms enthält.

CHAIN-Simulation:

```
Comm$=SPACE$(128)
BMOVE BASEPAGE+128,VARPTR(Comm$),128
~SHEL_WRITE(1,1,1,Comm$,"Programm.Prg")
```

18.13 Window-Manager

WIND_CALC() — Fensterkoordinaten und -maße ermitteln

Back=WIND_CALC(Modus,Attribut,Inx,Iny,Inb,Inh,...
...< <Outx,< <Outy,< <Outb,< <Outh)

Ermittelt aus bekannten Koordinaten und Maßen eines Fensters (Arbeitsbereich oder Gesamtfläche) die jeweils anderen Koordinaten und Maße desselben Fensters (Arbeitsbereich -> Gesamtfläche/Gesamtfläche -> Arbeitsbereich).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt
Modus	Funktionsmodus: 0 = Gesamtfläche des Fensters berechnen. 1 = Arbeitsbereich des Fensters berechnen.
Attribut	Spezifikation der Fenster-Randelemente, s. WINDTAB -> Attribut.
Inx, Iny, Inb, Inh:	Koordinaten der jeweils bekannten Fläche: Modus = 0 -> Arbeitsbereich Modus = 1 -> Gesamtfensterfläche
<<	Die Rückgabe-Variablen Outx, Outy, Outb und Outh enthalten nach Abschluß die Koordinaten und Maße der zu berechnenden Fläche. Modus = 0 <- Gesamtfensterfläche Modus = 1 <- Arbeitsbereich

WIND_CLOSE() — Fenster schließen (CLOSEW)

Back=WIND_CLOSE(Handle)

Ein mit WIND_OPEN geöffnetes GEM-Window wird vom Bildschirm gelöscht (vgl. CLOSEW). Das Handle bleibt jedoch noch für dieses Fenster reserviert und es kann durch WIND_OPEN wieder dargestellt werden. Bei Öffnung des Fenster durch WIND_OPEN wird FORM_DIAL(0) und bei Schließung FORM_DIAL(3) automatisch aufgerufen. Betreffend der Hintergrund-Restauration siehe Erläuterungen zu FORM_DIAL.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle des zu schließenden Fensters.

WIND_CREATE() Fenster definieren/anmelden

Back=WIND_CREATE(Attribut,Max_xl,Max_yo,Max_br,Max_ho)

Es wird Speicher für ein GEM-Window reserviert und das Fenster mit bestimmten Attributen und Maximalausdehnung angemeldet. AES liefert nach Abschluß das Window-Handle, mit welchem dieses Fenster ab jetzt ansprechbar ist.

Back	< 1 = Keine Window-Handle mehr verfügbar. > 0 = Handle des neu angemeldeten Fensters.
Attribut	Spezifikation der Fenster-Randelemente, s. WINDTAB -> Attribut.
Max_xl	X-Grenzkoordinate der maximalen Window-Ausdehnung.
Max_yo	Y-Grenzkoordinate der maximalen Window-Ausdehnung.
Max_br	Breite des Fensters in seiner maximalen Ausdehnung.
Max_ho	Höhe des Fensters in seiner maximalen Ausdehnung.

WIND_DELETE() Fenster löschen/abmelden

Back=WIND_DELETE(Handle)

Ein mit WIND_CREATE angemeldetes GEM-Window wird aus der Fensterliste gestrichen, der Speicherplatz und das Handle wieder freigegeben. Es kann erst durch WIND_OPEN wieder dargestellt werden, wenn das entsprechende Handle durch WIND_CREATE erneut ausgegeben wurde. Das hiermit abgemeldete Fenster sollte ggfs. vorher vom Bildschirm gelöscht werden.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle des abzumeldenden Fensters.

WIND_FIND() Fenster-Handle ermitteln

Back=WIND_FIND(Xpos,Ypos)

Liefert das Window-Handle des Fensters, das sich an der angegebenen Bildschirmposition befindet.

Back	0 = Es befindet sich kein GEM-Fenster an der angegebenen Position. > 0 = Handle des gefundenen Fensters.
Xpos	X-Koordinate der betreffenden Bildschirmposition.
Ypos	Y-Koordinate der betreffenden Bildschirmposition.

WIND_GET() — Fenster-Attribute ermitteln

Back=WIND_GET(Handle,Modus,< <Get1,< <Get2,< <Get3,< <Get4)

Lieferung verschiedener Auskünfte über GEM-Windows.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle des betreffenden Fensters.
Modus	Funktionsmodus. Gibt an, welche Auskünfte gewünscht werden. Abhängig vom gewählten Modus werden von GEM in den Rückgabevariablen Get1, Get2, Get3 und Get4 Daten mit wechselnder Bedeutung abgelegt.
4	Koordinaten und Maße des Arbeitsbereichs liefern: Get1 : X-Koordinate der linken oberen Ecke Get2 : Y-Koordinate der linken oberen Ecke Get3 : Breite des Bereichs Get1 : Höhe des Bereichs
5	Koord./Maße des aktuellen Gesamtfensters liefern. Bedeutung von Get1, Get2, Get3, Get4 wie bei 4.
6	Koord./Maße des vorher offenen Gesamtfensters liefern. Bedeutung von Get1, Get2, Get3, Get4 wie bei 4.
7	Koord./Maße der größtmöglichen Ausdehnung liefern. Bedeutung von Get1, Get2, Get3, Get4 wie bei 4.
8	Stellung des Horizontalschiebers relativ z. umfassenden Rahmen liefern: Get1 : 1 = Ganz links ; 1000 = Ganz rechts Get2, Get3, Get4 bleiben unverändert
9	Stellung des Vertikalschiebers relativ zum umfassenden Rahmen liefern: Get1 : 1 = Ganz oben ; 1000 = Ganz unten Get2, Get3, Get4 bleiben unverändert
10	Handle des aktuell geöffneten Fensters liefern: Get1 : Handle Get2, Get3, Get4 bleiben unverändert
11	Koord./Maße des ersten Rechtecks in der Rechteckliste des aktuellen Fensters liefern. Bedeutung von Get1, Get2, Get3, Get4 wie bei 4.
12	Koord./Maße des nächsten Rechtecks in der Rechteckliste des aktuellen Fensters liefern. Bedeutung von Get1, Get2, Get3, Get4 wie bei 4.
15	Größe des Horizontalschiebers relativ zur Größe des umfassenden Rahmens liefern: Get1: -1 = Minimalgröße (Quadrat) 1 = Ganz klein 1000 = Schieber:Rahmen = 1:1 Get2, Get3, Get4 bleiben unverändert.
16	Größe des Vertikalschiebers relativ zur Größe des umfassenden Rahmens liefern. Bedeutung von Get1 wie bei 15. Get2, Get3, Get4 bleiben unverändert

WIND_OPEN() — Fenster darstellen (OPENW)

Back=WIND_OPEN(Handle,Xpos,Ypos,Breite,Höhe)

Ein mit WIND_CREATE angemeldetes GEM-Window wird mit den vorgegebenen Attributen gezeichnet und reagiert ggfs. auf Anwendereinflüsse (vgl. OPENW).

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle des zu öffnenden Fensters.
Xpos	Beim Öffnen gewünschte X-Koordinate des Fensters.
Ypos	Beim Öffnen gewünschte Y-Koordinate des Fensters.
Breite	Beim Öffnen gewünschte Breite des Fensters.
Höhe	Beim Öffnen gewünschte Höhe des Fensters.

WIND_SET() — Fenster-Attribute ändern

Back=WIND_SET(Handle,Modus,Set1,Set2,Set3,Set4)

Änderung verschiedener Komponenten eines GEM-Windows.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Handle	Handle des betreffenden Fensters.
Modus	Funktionsmodus. Gibt an, welche Komponenten geändert werden sollen. Abhängig vom gewählten Modus werden von GEM die Daten in den Parametern Set1, Set2, Set3 und Set4 unterschiedlich interpretiert.
1	Attribute des angegebenen Fensters ändern. Set1: Neue Spezifikation der Fenster-Randelemente. Siehe dazu unter WINDTAB -> Attribut. Set2, Set3, Set4 bleiben unberücksichtigt
2	Fenster-Titel NAME ändern (vgl. TITLEW). Set1 : HI-Word eines Zeigers auf den neuen Titel. Set2 : LO-Word eines Zeigers auf den neuen Titel. Organisation des Strings: Byte 1: 1. Zeichen Byte 2: 2. Zeichen ... Byte n: letztes Zeichen Als Abschluß des Strings werden zwei Null-Bytes erwartet. Set3, Set4 bleiben unberücksichtigt
3	Infozeile INFO ändern (vgl. INFOW). Bedeutung von Set1, Set2 wie bei 2. Set3, Set4 bleiben unberücksichtigt.
5	Fensterkoordinaten und -größe ändern. Set1: Neue X-Koordinate des Fensters. Set2: Neue Y-Koordinate des Fensters. Set3: Neue Breite des Fensters. Set4: Neue Höhe des Fensters.
8	Relative Position des Horizontalschiebers ändern. Set1: 1 = Ganz links; 1000 = Ganz rechts. Set2, Set3, Set4 bleiben unberücksichtigt.
9	Relative Position des Vertikalschiebers ändern. Set1: 1 = Ganz oben; 1000 = Ganz unten Set2, Set3, Set4 bleiben unberücksichtigt.

10	Angegebenes Fenster aktivieren (vgl. TOPW). Set1, Set2, Set3, Set4 bleiben unberücksichtigt.
14	Objektbaum für neues Desktop initialisieren: Set1: HI-Word Set2: LO-Word eines Zeigers auf die Startadresse des Objektbaumes, in dem das neue Desktop beschrieben ist. Set3: Nummer des Objektes, mit dem begonnen werden soll. Set4: Bleibt unberücksichtigt. Wird in Set1, Set2 und Set3 jeweils eine Null übergeben, wird damit das Standard-Desktop installiert.
15	Größe des Horizontalschiebers relativ zur Größe des umfassenden Rahmens bestimmen. Set1: -1 = Minimalgröße (Quadrat) 1 = Ganz klein 1000 = Schieber:Rahmen = 1:1 Set2, Set3, Set4 bleiben unberücksichtigt.
16	Größe des Vertikalschiebers relativ zur Größe des umfassenden Rahmens bestimmen. Bedeutung von Set1 wie bei 15. Set2, Set3, Set4 bleiben unverändert.

WIND_UPDATE() Fenster-Darstellung kontrollieren

Back=WIND_UPDATE(Modus)

Die Funktion hat zum einen die Aufgabe, GEM mitzuteilen, daß das Programm gerade mit dem Neuaufbau eines Bildschirmbereichs beschäftigt ist (Window-Redraw) und dabei durch andere Applikationen (Accessories) nicht gestört werden möchte. Dazu wird die momentane Rechteckliste des aktuellen Fensters freigegeben ("eingefroren") und alle Menü-Zugriffe gesperrt.

Als Umkehrfunktion kann man GEM natürlich auch mitteilen, daß Menü-Zugriffe wieder zugelassen werden und die Rechteckliste "dynamisiert" werden soll. D.h., die Rechtecklisten der Fenster werden bei Window-Bewegungen und Formularaufrufen ständig dem neuesten Stand angepaßt. Zum zweiten bietet diese Funktion die Möglichkeit, GEM die Kontrolle der Mausaktivitäten zu entziehen. Die Mauskontrolle liegt in diesem Fall ausschließlich bei der aktuellen Applikation (also bei Ihrem Programm).

AES meldet dann weder Zugriffe auf Fensterbedienungselemente (FULLER/CLOSER/MOVER etc.), noch Zugriffe auf ein evtl. vorhandenes Pull-Down-Menü.

Hier ist wichtig zu wissen, daß GFA-BASIC während der Menü-Sperre trotzdem die Funktionsfähigkeit von ON MENU BUTTON, ON MENU KEY und ON MENU IBOX/OBOX gewährleistet.

Back	0 = Fehler; <> 0 = Funktion korrekt ausgeführt.
Modus	Funktionsmodus: 0 = Bildschirmaufbau ist fertig. Rechtecklisten freigeben. 1 = Bildschirmaufbau hat begonnen. Rechtecklisten sperren. 2 = Mauskontrolle bei AES (Menü/Windows sind aktiv). 3 = Mauskontrolle beim Programm (Menü/Windows sind inaktiv).

18.14 Beispiele zu FORM_x, OBJC_x, RSRC_x und MENU_x

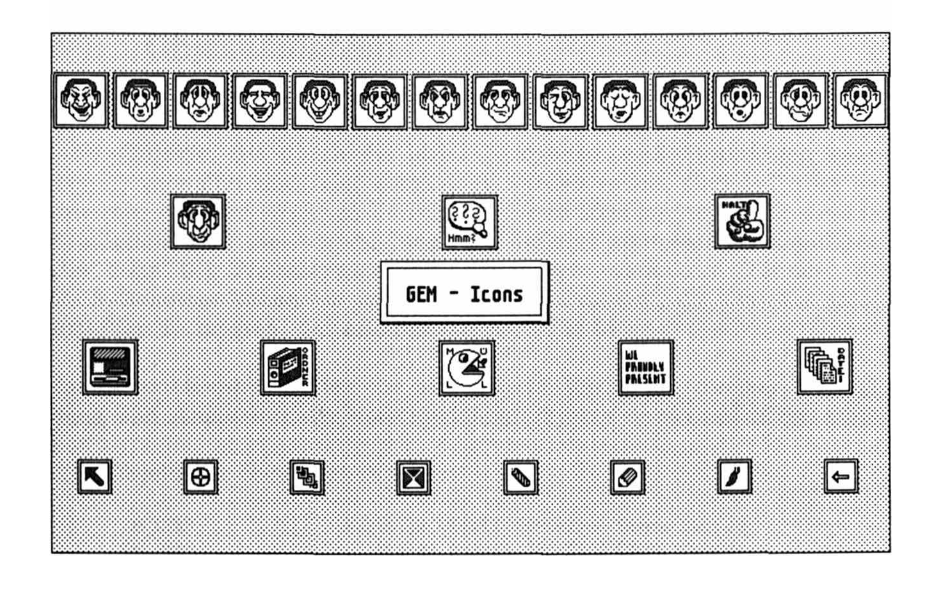
```
' FORM_xxxx -/OBJC_xxxx -/RSRC_xxxx - Demo
' ***********************************************
LET tree1&=0       ! RSRC_TREE           -------.
LET t1_txt&=1      ! Text-Objekt in Baum 0       | Objekt-Indizes
LET t1_ok&=3       ! OKAY-Objekt in Baum 0       | für 1. Formular
LET t1_can&=4      ! Cancel-Objekt in Baum 0 -'
'
LET tree2&=1       ! RSRC_TREE           -------.
LET t2_txt&=1      ! Text-Objekt in Baum 1       |
LET t2_but1&=4     ! Objekt in Baum 1            |
LET t2_but2&=5     ! Objekt in Baum 1            |
LET t2_but3&=6     ! Objekt in Baum 1            |
LET t2_but4&=7     ! Objekt in Baum 1            |
LET t2_but5&=8     ! Objekt in Baum 1            | Objekt-Indizes
LET t2_but6&=9     ! Objekt in Baum 1            | für 2. Formular
LET t2_but7&=10    ! Objekt in Baum 1            |
LET t2_but8&=11    ! Objekt in Baum 1            |
LET t2_but9&=12    ! Objekt in Baum 1            |
LET t2_but10&=13   ! Objekt in Baum 1            |
LET t2_ok&=14      ! OKAY-Objekt in Baum 1       |
LET t2_can&=15     ! Cancel-Objekt in Baum 1 -'
'
RESERVE FRE(0)-10000              ! Speicher für Resource freigeben
~RSRC_LOAD("formular.rsc")        ! Resource laden
~RSRC_GADDR(0,tree1&,form1%)      ! Startadresse von Baum 0 holen
~FORM_CENTER(form1%,x,y,w,h)      ! Formular 0 zentrieren
~RSRC_GADDR(0,tree2&,form2%)      ! Startadresse von Baum 1 holen
~FORM_CENTER(form2%,x2,y2,w2,h2)  ! Formular 1 zentrieren
CHAR{{OB_SPEC(form2%,t2_txt&)}}="00" ! Textobjekt 2 vorbelegen
'
DEFFILL ,2,4                      ! Hintergrund vorbereiten, da
PBOX 120,85,515,325               ! FORM_DIAL() graue Flecken hinterläßt
PRINT " Baum 0 mit linker Maustaste"
PRINT " Baum 1 mit rechter Maustaste"
PRINT " Abbruch mit beliebiger <Taste>"
DO
  EXIT IF INKEY$<>""              ! Exit bei beliebiger <Taste>
  IF MOUSEK=1                     ! Linke Maustaste gedrückt?
    ~FORM_DIAL(1,0,0,0,0,x,y,w,h)   ! GROWBOX produzieren
    ~OBJC_DRAW(form1%,0,3,x,y,w,h)  ! Formular 0 zeichnen
    ~FORM_DIAL(0,0,0,0,0,x,y,w,h)   ! Restore-Rechteck speichern
    ret_ob&=FORM_DO(form1%,t1_txt&) ! Formular-Bedienung übernehmen
    ~FORM_DIAL(3,0,0,0,0,x,y,w,h)   ! Formular überdecken
    ~FORM_DIAL(2,0,0,0,0,x,y,w,h)   ! SHRINKBOX produzieren
    ~OBJC_CHANGE(form1%,ret_ob&,0,x,y,w,h,0,0) ! Status des EXIT-
    '                               ! Objekts wieder rücksetzen
    PRINT AT(1,10);"Ihr Alter: ";CHAR{{OB_SPEC(form1%,t1_txt&)}}'
```

```
    '                                ! Text-Objekt-Inhalt lesen
  ENDIF
  IF MOUSEK=2                        ! Rechte Maustaste gedrückt?
    FOR objc&=t2_but1& TO t2_but10& ! Alle "Alter"-Buttons durchgehen
    EXIT IF AND(OB_STATE(form2%,objc&),1) ! SELECTED-Status gesetzt?
  NEXT objc&
  old_objc&=objc&                   ! Alten Status merken
  old_str$=CHAR{{OB_SPEC(form2%,t2_txt&)}} ! Alten Text merken
  ~FORM_DIAL(1,0,0,0,0,x2,y2,w2,h2) ! GROWBOX produzieren
  ~OBJC_DRAW(form2%,0,3,x2,y2,w2,h2)! Formular 1 darstellen
  ~FORM_DIAL(0,0,0,0,0,x2,y2,w2,h2) ! Restore-Rechteck speichern
  REPEAT                            ! Wiederhole solange, bis <----------.
    ret_ob&=FORM_DO(form2%,0)  ! Formular-Bedienung übernehmen           |
    EXIT IF ret_ob&=t2_ok& OR ret_ob&=t2_can& ! Exit, wenn OKAY         |
    '                               ! oder Cancel gewählt wurde          |
    FOR objc&=t2_but1& TO t2_but10& ! Alle Alter-Buttons                 |
      EXIT IF BTST(OB_STATE(form2%,objc&),0) ! SELECTED-Bit gesetzt      |
    NEXT objc&                                                           |
    string$=STR$((objc&-t2_but1&)*5+10) ! Gewählten Alter-Button         |
    '                                   ! in String umwandeln            |
    CHAR{{OB_SPEC(form2%,t2_txt&)}}=string$ ! evtl. veränderten          |
    '                                       ! Alter-String eintragen     |
    ~OBJC_DRAW(form2%,t2_txt&,1,x2,y2,w2,h2) ! Kindebene neu zeichnen|
  UNTIL ret_ob&=t2_ok& OR ret_ob&=t2_can& ! OKAY o. Cancel gewählt  >'
  ~FORM_DIAL(3,0,0,0,0,x2,y2,w2,h2) ! Formular überdecken
  ~FORM_DIAL(2,0,0,0,0,x2,y2,w2,h2) ! SHRINKBOX produzieren
  ~OBJC_CHANGE(form2%,ret_ob&,0,x,y,w,h,0,0) ! Status zurücksetzen
  IF ret_ob&=t2_can&                ! "Cancel" gewählt
    ~OBJC_CHANGE(form2%,objc&,0,x2,y2,w2,h2,0,0) ! Status des zuletzt
    '                               ! gewählten Buttons wieder auf Null
    ~OBJC_CHANGE(form2%,old_objc&,0,x2,y2,w2,h2,1,0) ! Status des
    '                               ! vorher "aktiven" Buttons wieder
    '                               ! auf SELECTED setzen
    CHAR{{OB_SPEC(form2%,t2_txt&)}}=old_str$ ! Text restaurieren
  ENDIF
ENDIF
LOOP
~RSRC_FREE()                        ! Resource löschen
RESERVE FRE(0)+10000                ! Speicher restaurieren
' MENU_xxxx - Demo
' ******************
LET menu_tree%=0 ! RSRC_TREE
LET desk%=3      ! Objekt in Baum 0 (Menütitel)
LET tips%=4      ! Objekt in Baum 0 (Menütitel)
LET icons%=5     ! Objekt in Baum 0 (Menütitel)
LET smily%=6     ! Objekt in Baum 0 (Menütitel)
'
LET myprog%=9    ! Objekt in Baum 0
LET t1%=19       ! Objekt in Baum 0
LET t2%=20       ! Objekt in Baum 0
LET t3%=21       ! Objekt in Baum 0
LET t4%=22       ! Objekt in Baum 0
LET t5%=23       ! Objekt in Baum 0
LET q.uit%=25    ! Objekt in Baum 0
LET ausgang%=37  ! Objekt in Baum 0
'
RESERVE FRE(0)-10000          ! Speicher für Resource freigeben
IF RSRC_LOAD("menu.rsc")      ! Resource laden
  ~RSRC_GADDR(0,menu_tree%,menu%) ! Menübaum-Adresse holen
```

```
    ON MENU GOSUB do_menu          ! Verzweigung bei Menü-Ereignissen
    ~MENU_BAR(menu%,1)             ! Menü aktivieren
    DO                             ! Überwachungs-Schleife
      EXIT IF e.flag!              ! Abbruch, wenn "Ausgang" gewählt
      ON MENU                      ! Events feststellen
    LOOP
    ~MENU_BAR(menu%,0)             ! Menü deaktivieren
    ~RSRC_FREE()                   ! Resource löschen
  ELSE
    ALERT 1,"'MENU.RSC' konnte nicht|geladen werden !",1,"Return",b%
  ENDIF
  RESERVE FRE(0)+10000             ! Speicher restaurieren
  '
  PROCEDURE do_menu
    PRINT AT(11,11);"Menüpunkt-Index :"'MENU(5)'
    IF MENU(5)=myprog%             ! Menüpunkt im Desk-Menü gewählt?
      ~MENU_TEXT(menu%,myprog%,"Dein Programm") ! Text austauschen
    ENDIF
    IF MENU(5)=ausgang%            ! "Ausgang" gewählt
      e.flag!=TRUE                 ! Abbruch-Flag setzen
    ENDIF
    IF MENU(4)=tips%               ! Tips-Menü war offen?
      IF MENU(5)=q.uit%            ! Darin Punkt "Ende" gewählt?
        FOR n%=t1% TO t5%          ! Alle anderen Menüpunkte
          ~MENU_IENABLE(menu%,n%,1) ! wieder wählbar machen
        NEXT n%
      ELSE                         ! Sonst...
        ~MENU_IENABLE(menu%,MENU(5),0) ! ...den gewählten Punkt..
      ENDIF                        ! ...unwählbar machen
    ENDIF
    ~MENU_TNORMAL(menu%,MENU(4),1)   ! Menütitel "schwarz auf weiß"
  RETURN
```

Das Interessante an diesem Beispiel ist, wie die Grafiken in das Menü gelangt sind. Wie man grundsätzlich Grafiken in das RCS importiert, habe ich weiter oben in der RCS-Beschreibung schon angedeutet. Da aber im RCS für einen Menü-Typ keine Icons und Images vorgesehen sind, können Sie aus der Menü-Part-Box auch keine solchen in einen Menübaum hereinholen.

Hier gibt es den Trick, nachdem das Menü als solches erstellt ist, es durch die Menü-Option "Type" im Menütitel "Options" einfach in einen Dialog-Typ umzuwandeln (erst den Baum schließen) und dann die Icons im neu geöffneten Dialog-Baum zu plazieren. Nachdem Sie die gewünschten Icons auf die unten beschriebene Art und Weise in das RCS "gemergt" und im Baum eingebaut haben, schließen Sie den Baum wieder, gehen wieder in das Menü "Options", ändern durch "Type" erneut den Dialog-Typ in einen Menü-Typ um und haben so die Icons in das Menü geschmuggelt.


Hmm?
HALT
GEM - Icons
ORDNER
WE PROUDLY PRESENT
DATEI

19. Objektvariablen

AES-Objekte, die durch die objektbezogenen AES-Funktionen bearbeitet, verwaltet und/oder dargestellt werden sollen, sind auf eine bestimmte Art und Weise zu definieren. Diese sogenannte Objektstruktur ermöglicht erst die Verwendung von äußerst komplexen Formularen, die dann die Programmierarbeit wesentlich erleichtern können. Als RCS-Benutzer haben Sie jedoch nicht das Problem, die Objekte selbst erstellen zu müssen, das erledigt das RCS für Sie.

In diesem Kapitel finden Sie alle Objekt-Variablen, die zur Verwaltung und Anpassung der verschiedenen Objekte direkt aufrufbar sind.

Ein Objekt hat folgenden Aufbau:

```
OB_NEXT        Nächstes Objekt derselben Ebene (Word)
OB_HEAD           Erstes Objekt der Kind-Ebene (Word)
OB_TAIL          Letztes Objekt der Kind-Ebene (Word)
OB_TYPE                             Objekt-Typ (Word)
OB_FLAGS                     Objekt-Attribute (Word)
OB_STATE                        Objekt-Status (Word)
OB_SPEC             Adresse der Unter-Struktur (Long)
OB_X               Relative Objekt-X-Koordinate (Word)
OB_Y               Relative Objekt-Y-Koordinate (Word)
OB_W                            Objektbreite (Word)
OB_H                             Objekthöhe (Word)
```

Den Aufbau der Unter-Strukturen (TEDINFO, BITBLK etc.) finden Sie unter 19.1 "GEM-Datenstrukturen" beschrieben.

Alle Objekt-Variablen sind zweifach indiziert, wodurch das betreffende Objekt genau spezifiziert werden kann.

Parameter 1 -> Adresse
Adresse des Objektbaumes, dem das betreffende Objekt angehört.

Parameter 2 -> Objekt
Nummer des betreffenden Objektes innerhalb des mit Adresse bestimmten Baumes. Die erste (oberste) Ebene hat immer Nummer 0.

Der Ausdruck Wert hat in den Syntax-Beschreibungen folgende Bedeutung:

In Zuweisungen
Wert, der an die Variable übergeben werden soll.

In Nachfragen
Wert, der momentan in der Variablen enthalten ist.

OB_ADR() — Objekt-Adresse (Long)

OB_ADR(Adresse,Objekt)=Wert **-> Zuweisung**
Wert=OB_ADR(Adresse,Objekt) **-> Nachfrage**

- *Kein Bestandteil der Objektstruktur* -
Liefert in Wert den Byte-Offset des betreffenden Objektes zur angegebenen Start-Adresse des übergeordneten Objektbaums. Durch Zuweisung kann die Reihenfolge der Objekte innerhalb der Baumstruktur bestimmt werden bzw. den Objekten ein willkürlicher Offset zugewiesen werden.

OB_NEXT() — Nächstes Objekt derselben Ebene lesen (Word)
OB_NEXT()= {Ob)= } — ... schreiben (Word)

Wert=OB_NEXT(Adresse,Objekt) **-> Nachfrage**
OB_NEXT(Adresse,Objekt)=Wert **-> Zuweisung**

Enthält die Nummer des nächsten Objektes auf der Ebene, welcher das angegebene Objekt selbst angehört. Ist das angegebene Objekt das letzte Objekt der Ebene, zeigt OB_NEXT auf das ihm übergeordnete Wurzelobjekt (Eltern-Generation) zurück. Ist es selbst das oberste Wurzelobjekt, enthält OB_NEXT eine -1.

OB_HEAD() — Erstes Objekt der Kind-Ebene lesen (Word)
OB_HEAD()= {Ob_h)= } — ... schreiben (Word)

Wert=OB_HEAD(Adresse,Objekt) **-> Nachfrage**
OB_HEAD(Adresse,Objekt)=Wert **-> Zuweisung**

Enthält die Nummer des ersten Objektes der nächsttieferen Ebene (1. Kind). Ist keine Kind-Ebene vorhanden, enthält OB_HEAD eine -1.

OB_TAIL() — Letztes Objekt der Kind-Ebene lesen (Word)
OB_TAIL()= { Ob_t)= } — ... schreiben (Word)

Wert=OB_TAIL(Adresse,Objekt) -> **Nachfrage**
OB_TAIL(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält die Nummer des letzen Objektes der nächsttieferen Ebene (n. Kind). Ist keine Kind-Ebene vorhanden, enthält OB_TAIL eine -1.

OB_TYPE() — Objekt-Typ lesen (Word)
OB_TYPE()= { Ob_ty)= } — ... schreiben (Word)

Wert=OB_TYPE(Adresse,Objekt) -> **Nachfrage**
OB_TYPE(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält den Typen-Index des betreffenden Objektes. Die Bedeutung der Objekt-Variable OB_SPEC wechselt bei den verschiedenen Objekttypen.

Index	Bezeichnung	OB_SPEC
20	G_BOX Rechteck.	BOXINFO (32-Bit-Code)
21	G_TEXT Variabler Grafiktext.	Zeiger auf TEDINFO
22	G_BOXTEXT Text in einem Rechteck.	Zeiger auf TEDINFO
23	G_IMAGE Bitraster-Zeichnung.	Zeiger auf BITBLK
24	G_PROGDEF Maschinen-Routine zur eigenen Objekt-Darstellung.	Zeiger auf APPLBLK
25	G_IBOX Rechteck ohne Rahmen und Füllung. Als unsichtbares Wurzelobjekt geeignet.	BOXINFO (32-Bit-Code)
26	G_BUTTON Grafiktext. Zentriert in einem Rechteck (Abschluß durch Rechteck).	Zeiger auf C-Text
27	G_BOXCHAR Einzelner Buchstabe. Zentriert in einem Rechteck.	BOXINFO (32-Bit-Code)
28	G_STRING Grafiktext in Standard-Font (Abschluß durch Rechteck) z.B. Pull-Down-Menütext.	Zeiger auf C-Text
29	G_FTEXT Formatierter Grafiktext (siehe TEDINFO).	Zeiger auf TEDINFO
30	G_FBOXTEXT Formatierter Grafiktext in einem Rechteck (siehe TEDINFO).	Zeiger auf TEDINFO
31	G_ICON Maskierte Bit-Raster-Zeichnung mit Text-Integration.	Zeiger auf ICONBLK
32	G_TITLE Grafiktext für Pull-Down-Menü-Titel.	Zeiger auf C-Text

OB_FLAGS() — Objekt-Attribute lesen (Word)
OB_FLAGS()= {Ob_f)=} — ... schreiben (Word)

Wert=OB_FLAGS(Adresse,Objekt) -> **Nachfrage**
OB_FLAGS(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält einen 9-Bit-Vektor, der angibt, mit welchen Attributen das Objekt ausgestattet sein soll. Ein gesetztes Bit bedeutet, daß dem Objekt das betreffende Attribut zugeordnet wird. Ist kein Bit gesetzt, ist es ein sog. NONE-Objekt. Zur Selektion der einzelnen Bits siehe unter BTST/BCHG/BSET/BCLR.

Bit	Name	Erläuterung
0	SELECTABLE	Wählbares Objekt. Bei Mausklick auf Objekt wird es invers gezeichnet.
1	DEFAULT	Objekt, das auch durch <Return> angewählt werden kann. Üblicherweise auch EXIT-Objekt. Pro Formular (Baum) ist nur ein DEFAULT-Objekt erlaubt.
2	EXIT	Auswahl dieses Objektes führt dazu, daß Maus- und Tastaturkontrolle wieder an das Programm zurückgegeben werden (Rückkehr vom FORM_DO-Aufruf).
3	EDITABLE	Das betreffende Objekt ist in irgendeiner Weise veränderbar (üblich: Texteingabe).
4	RBUTTON	Objekt innerhalb einer Objektgruppe (Objekte einer Ebene). Wird ein Objekt dieser Gruppe angewählt, wird ein evtl. schon aktiviertes Objekt derselben Gruppe wieder deaktiviert (normal). Innerhalb einer solchen Gruppe kann immer nur ein Objekt aktiviert (invers) werden.
5	LASTOB	Hierarchisch letztes Objekt des aktuellen Objektbaumes. Pro Formular (Baum) ist nur ein LASTOB-Objekt erlaubt.
6	TOUCHEXIT	Gleiche Funktion wie EXIT. Im Gegensatz zu EXIT muß die Maustaste jedoch nicht erst losgelassen werden, um das Formular zu verlassen.
7	HIDETREE	Kennzeichnet ein Wurzelobjekt, das inkl. sämtlicher Nachkommen (Unterebenen) von OBJC_DRAW, OBJC_FIND, FORM_DO etc. "übersehen" wird. Die Objekte dieser Ebenen werden also weder dargestellt, noch sind sie im Formular wählbar.

8	INDIRECT	Die Objektspezifikation des hiermit markierten Objektes ist nicht der Wert selbst, sondern Zeiger darauf.

OB_STATE() — Objekt-Status lesen (Word)
OB_STATE()= {Ob_s)=} — ... schreiben (Word)

Wert=OB_STATE(Adresse,Objekt) **-> Nachfrage**
OB_STATE(Adresse,Objekt)=Wert **-> Zuweisung**

Enthält einen 6-Bit-Vektor, durch welchen die Art der grafischen Darstellung eines Objektes bestimmt werden kann. Ist kein Bit gesetzt, ist es ein sog. NORMAL-Objekt. Zur Selektion der einzelnen Bits siehe unter BTST/BCHG/BSET/BCLR.

Bit	Name	Erläuterung
0	SELECTED	Das betreffende Objekt wird invers gezeichnet, um zu signalisieren, daß der dadurch repräsentierte Auswahlpunkt aktiviert ist. Dieser Status wird einem SELECTABLE-Objekt auch durch FORM_DO() zugewiesen, sobald es innerhalb eines Formulars vom Anwender angeklickt wurde.
1	CROSSED	Das Objekt wird mit zwei Linien diagonal "durchkreuzt". Eignet sich vorwiegend für G_??BOX??-Typen, um es als "markiert" zu kennzeichnen.
2	CHECKED	In das Objekt wird ein Häkchen gezeichnet, um es als "erledigt" zu kennzeichnen (z.B. bei Menüzeilentext).
3	DISABLED	Das Objekt wird hell (lightend) gezeichnet, um es als "unwählbar" zu kennzeichnen (dazu auch Bit 0 von OB_FLAGS ausschalten).
4	OUTLINED	Das Objekt wird mit einem dicken und einem weiteren, dünnen Rahmen eingerahmt, um ggfs. seine Priorität zu unterstreichen (evtl. bei Rahmen-, bzw. Wurzelobjekten).
5	SHADOWED	Das Objekt wird mit einem nach rechts unten fallenden Schatten ausgestattet. Die Schattentiefe richtet sich nach der Randstärke des Objektes (optischer Effekt).

OB_SPEC() — Adresse der Unter-Struktur lesen (Long)
OB_SPEC()= { Ob_sp)= } — ... schreiben (Long)

Wert=OB_SPEC(Adresse,Objekt) -> **Nachfrage**
OB_SPEC(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält einen Wert, dessen Bedeutung je nach OB_TYPE variiert. Mögliche Bedeutungen sind TEDINFO-, BITBLK-, ICONBLK-, APPLBLK- und String-Zeiger oder BOXINFO-Wert (s. OB_TYPE).

OB_X() — Relative Objekt-X-Koordinate (Word)

Wert=OB_X(Adresse,Objekt) -> **Nachfrage**
OB_X(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält die X-Koordinate des angegebenen Objekts. Bei selbstdefinierten Objekten mit zeichenorientierten Koordinaten sind diese vor Objekt-Aufruf umzurechnen (siehe RSRC_OBFIX). Beim Objekt der obersten Ebene (Wurzelobjekt) bezieht sich die Angabe der Koordinaten immer auf den absoluten Bildschirmnullpunkt. Bei allen Unterobjekten stehen die Koordinatenangaben immer in Relation zum Nullpunkt (obere linke Ecke) des übergeordneten (Eltern-)Objektes. Daraus ergibt sich, daß übergeordnete Objekte immer in Form eines koordinatenbehafteten Rechtecks zu definieren sind.

Unterobjekt-Koordinaten sind immer so anzugeben, daß das jeweilige Unterobjekt immer vollständig innerhalb des Rechtecks des Eltern-Objektes liegt.

OB_Y() — Relative Objekt-Y-Koordinate (Word)

Wert=OB_Y(Adresse,Objekt) -> **Nachfrage**
OB_Y(Adresse,Objekt)=Wert -> **Zuweisung**

Enthält die Y-Koordinate des angegebenen Objekts. Siehe Erläuterungen zu OB_X.

OB_W() — Objektbreite (Word)

Wert=OB_W(Adresse,Objekt) -> Nachfrage
OB_W(Adresse,Objekt)=Wert -> Zuweisung

Enthält die Breite des angegebenen Objekts. Siehe Erläuterungen zu OB_X.

OB_H() — Objekthöhe (Word)

Wert=OB_H(Adresse,Objekt) -> Nachfrage
OB_H(Adresse,Objekt)=Wert -> Zuweisung

Enthält die Höhe des angegebenen Objekts. Siehe Erläuterungen zu OB_X.

19.1 GEM-Datenstrukturen

Zur Objektprogrammierung mit AES ist die Kenntnis über den Aufbau der Unter- und Unter-Unter-Strukturen unerläßlich. Bei den verschiedenen Objekttypen (siehe OB_TYPE) werden zur weiteren Spezifikation jeweils Zeiger auf eine entsprechende Unterstruktur erwartet, bzw. in den Fällen G_BOX, G_IBOX und G_BOXCHAR ein Long-Wert mit weiteren Informationen zur Boxbeschaffenheit (BOXINFO).

RSC-HEADER — RSC-File-Vorspann (38 Bytes)

Offset	Name/Bedeutung	Format
0	Reserviert (immer 1)	Word
2	Objekt-Feld-Position	Word
4	Tedinfo-Position	Word
6	Icon-Block-Position	Word
8	Bit-Block-Position	Word
10	Position freier Strings	Word
12	Position gebundener Strings	Word
14	Position der Image-Daten	Word
16	Position freier Images	Word
18	Objektbaum-Tabellenzeiger	Word
20	Objekt-Anzahl im RSC-File	Word
22	Baum-Anzahl im RSC-File	Word
24	Anzahl der TEDINFOs	Word
26	Anzahl der ICNBLKs	Word
28	Anzahl der BITBLKs	Word
30	Anzahl der Strings	Word
32	Anzahl der Images	Word

34	Länge der RSC-Datei	Word
36	End-Markierung (0)	Word

BOXINFO — 32-BIT-Code in OB_SPEC

Dieser Wert wird bei den Objekttypen G_BOX, G_IBOX und G_BOXCHAR in OB_SPEC erwartet.

Aufbau:

```
 ---------------------------------------------------------
 |          HI-Word            |          LO-Word          |
 |   HI-Byte    |   LO-Byte    |   HI-Byte   |   LO-Byte   |
 |======|======|======|======|======|======|======|======|
 | 7777 | 7777 | 6666 | 6666 | 5555 | 4444 | 3222 | 1111 |
 |======|======|======|======|======|======|======|======|
   ^                        ^   ^                      ^
  Bit 31                Bit 16 Bit 15                Bit 0
```

Die Ziffern haben dabei folgende Bedeutung:

1	Füllmusterfarbe (4 Bits = 0 bis 15)
2	Füllmuster (3 Bits = 0 bis 7) 0 = Ohne Füllmuster. 1-6 = Verschiedene Muster. 7 = Komplett mit Füllfarbe gefüllt.
3	Grafikmodus (1 Bit = 0 bis 1) 0 = Objekt replace zeichnen. 1 = Objekt transparent zeichnen.
4	Farbe f. Textzeichen (4 Bits = 0 bis 15). - Nur bei G_BOXCHAR-Objekten -
5	Farbe des Objektrahmens (4 Bits = 0 bis 15)
6	Rahmendicke (8 Bits = -127 bis 128) 0 = Kein Rahmen. 1 bis 127 = Rahmendicke nach innen. 128 bis 255 = Rahmendicke nach aussen. Komplementwert: Entspricht -127 bis -1.
7	ASCII für Textzeichen (8 Bits = 0 bis 255) - Nur bei G_BOXCHAR-Objekten -

TEDINFO — Textobjekt-Unterstruktur (28 Byte)

Enthält weitere Informationen über Text-Objekte vom Typ G_TEXT, G_BOXTEXT, G_FTEXT und G_FBOXTEXT (siehe OB_SPEC).

Die Struktur hat folgenden Aufbau:

Offset:	Name (Format)/Bedeutung:
0	te_ptext (Long): Zeiger auf C-Text, der die Vorgabe für die editierbaren Positionen in te_ptmplt enthält (z.B. 2000Hamburg). Ist das erste Zeichen ein @ (Klammeraffe/Commercial At), wird die komplette Textlänge als Space-String interpretiert.
4	te_ptmplt (Long) Zeiger auf einen C-Text mit der Textmaske, welche die String-Positionen bestimmt, in welche bei Edition vom Benutzer Zeichen eingegeben werden können. Diese Positionen sind durch einen Tiefstrich zu kennzeichnen, z.B.: PLZ:_____ Ort:________________. Bei Aufruf des Objektes wird te_ptext in diese Positionen als Vorgabe eingetragen. z.B.: PLZ:2000 Ort:Hamburg_______. Sind nur Ziffern zugelassen und es wird bei Edition ein alphanumerisches Zeichen eingegeben, springt der Cursor auf die nächste freie Eingabeposition hinter diesem Zeichen, sofern das Zeichen in der Schablone te_ptmplt enthalten ist.
8	te_pvalid (Long): Zeiger auf C-Text, in welchem die zur Eingabe zugelassenen Zeichen spezifiziert sind. Für jeden Tiefstrich in te_ptmplt ist hier das entsprechende Valid-Zeichen einzutragen, z.B.: 9999XXXXXXXXXXXX. Fehlerfreie Valid-Zeichen: 9 = Nur Ziffer (0 - 9) ist zugelassen. X = Jedes Zeichen ist zugelassen. Unzuverlässige Valid-Zeichen (Absturzgefahr): A = Nur Großbuchstabe oder Leerzeichen. a = Nur Groß-/Kleinbuchstabe oder Leerzeichen. N = Nur Ziffer, Großbuchstabe oder Leerzeichen. n = Wie N oder Kleinbuchstabe. F = File-Namen-Zeichen (Großbuchstabe oder _.:*?). p = Wie F, jedoch zusätzlich \. P = Wie p, jedoch nicht * oder ?
12	te_font (Word): Zeichensatz-Flag: 3 = Standard-Zeichensatz. 5 = Verkleinerter Zeichensatz.
14	te_resvd (Word): Für spätere TOS-Versionen reserviert.
16	te_just (Word): Flag für Textausrichtung: 0 = Linksbündig 1 = Rechtsbündig 2 = Zentriert
18	te_color (Word) 0 - 15 = Rahmenfarbe bei G_BOXTEXT und G_FBOXTEXT.
20	te_resvd2 (Word): Für spätere TOS-Versionen reserviert.
22	te_thickness (Word): Rahmendicke bei G_BOXTEXT/GFBOXTEXT: 0 = Kein Rahmen. 1 bis 127 = Rahmendicke nach innen. 128 bis 255 = Rahmendicke nach außen. Komplementwert: Entspricht -127 bis -1.
24	te_txtlen (Word): Länge des C-Textes (inkl. Null-Byte), dessen Zeiger in te_ptext angegeben ist.
26	te_tmplen (Word): Länge des C-Textes (inkl. Null-Byte), dessen Zeiger in te_ptmplt angegeben ist.

ICONBLK G_ICON-Unterstruktur (36 Byte)

Enthält weitere Informationen über Grafik-Objekte vom Typ G_ICON (siehe OB_SPEC). Ein G_ICON-Objekt besteht aus einer Hintergrund-Maske und der eigentlichen Grafik. Bei Aufruf wird zuerst die Maske gezeichnet und anschließend die Grafik auf die Maske gelegt. Zusätzlich hat man bei G_ICONs die Möglichkeit, einen Text-String und ein weiteres einzelnes Zeichen in die Grafik zu integrieren (siehe Laufwerksymbol im Desktop).

Die Struktur hat folgenden Aufbau:

Offset:	Name (Format)/Bedeutung:
0	ib_pmask (Long): Zeiger auf die Maskendaten im GEM-Bit-Raster-Modus (Aufbau wie GET/PUT-String ohne Header). Ein gesetztes Bit im Raster löscht den entsprechenden Bildschirmpunkt (= Hintergrundfarbe).
4	ib_pdata (Long): Zeiger auf die Grafikdaten im GEM-Bit-Raster-Modus (Aufbau wie GET/PUT-String ohne Header). Ein gesetztes Bit im Raster zeichnet den entsprechenden Bildschirmpunkt in der zugewiesenen Farbe.
8	ib_ptext (Long): Zeiger auf einen C-Text, der den in die Grafik zu integrierenden String enthält.
12	ib_col (Byte): Die unteren vier Bit enthalten die Farbe der Icon-Maske und die oberen vier Bit die Farbe des Icons.
13	ib_char (Byte): ASCII-Code eines einzelnen Zeichens, das in die Grafik integriert werden soll.
14	ib_xchar (Word): Pixelorientierte X-Position des einzelnen Zeichens, relativ zur linken oberen Ecke der G_ICON-Grafik.
16	ib_ychar (Word): Pixelorientierte Y-Position des einzelnen Zeichens, relativ zur linken oberen Ecke der G_ICON-Grafik.
18	ib_xicon (Word): Pixelorientierte X-Position der Grafik, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_X).
20	ib_yicon (Word): Pixelorientierte Y-Position der Grafik, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_Y).
22	ib_wicon (Word): Pixelorientierte Breite der Grafik (muß glatt durch 16 teilbar sein).
24	ib_hicon (Word): Pixelorientierte Höhe der Grafik.
26	ib_xtext (Word): Pixelorientierte X-Position des Text-Strings, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_X).
28	ib_ytext (Word): Pixelorientierte Y-Position des Text-Strings, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_Y).
30	ib_wtext (Word): Pixelorientierte Breite des Text-Strings.
32	ib_htext (Word): Pixelorientierte Höhe des Text-Strings.
34	ib_resvd (Word): Für spätere TOS-Versionen reserviert.

BITBLK G_IMAGE-Unterstruktur (14 Byte)

Enthält weitere Informationen über Grafik-Objekte vom Typ G_IMAGE (siehe OB_SPEC). Ein G_IMAGE-Objekt besteht lediglich aus einer GEM-Bit-Raster-Grafik ohne Maske (nur auf einfarbigem Hintergrund verwendbar - s. FORM_ALERT/ALERT-Symbole).

Die Struktur hat folgenden Aufbau:

Offset:	Name (Format)/Bedeutung:
0	bi_pdata (Long): Zeiger auf die Grafikdaten im GEM-Bit-Raster-Modus (Aufbau wie GET/PUT-String ohne Header). Ein gesetztes Bit im Raster zeichnet den entsprechenden Bildschirmpunkt in der zugewiesenen Farbe.
4	bi_wb (Word): Breite der Grafik in Bytes (muß glatt durch 2 teilbar sein).
6	bi_hl (Word): Pixelorientierte Höhe der Grafik.
8	bi_x (Word): Pixelorientierte X-Position der Grafik, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_X).
10	bi_y (Word): Pixelorientierte Y-Position der Grafik, relativ zur linken oberen Ecke des übergeordneten Objektes (siehe OB_Y).
12	bi_color (Word): 0-15 = Farbe der in ib_pdata gesetzten Grafikpunkte (0 = Hintergrundfarbe).

APPLBLK (USERBLK) G_PROGDEF-Unterstruktur (8 Byte)

Enthält einen Zeiger auf eine Maschinen- oder C-Routine, die bei Aufruf eines G_PROGDEF-Objekts (siehe OB_SPEC) ausgeführt wird.

Die Struktur hat folgenden Aufbau:

Offset:	Name (Format)/Bedeutung:
0	ub_code (Long): Zeiger auf eine Maschinen- oder C-Routine, die bei Objektaufruf angesprungen wird. Hiermit lassen sich beliebige Programme in die Objektdarstellung einbinden (z.B. um Sound- oder Grafikeffekte zu produzieren oder um die Formulardarstellung zu beeinflussen.
4	ub_parm (Long): Zeiger auf eine PARMBLK-Struktur, der als Parameter an die User-Routine übergeben wird.

PARMBLK

APPLBLK-Unterstruktur (30 Byte)

Enthält weitere Informationen über den aktuellen Baum, das aktuelle Objekt, sowie Angaben zur Objektdarstellung.

Die Struktur hat folgenden Aufbau:

Offset:	Name (Format)/Bedeutung:
0	pb_tree (Long): Adresse des Objektbaumes.
4	pb_obj (Word): Nummer des betreffenden Objektes.
6	pb_prevstate (Word): Vorheriger Objektstatus.
8	pb_currstate (Word): Aktueller Objektstatus.
10	pb_x (Word): Pixelorientierte X-Position des Objektes, relativ zur linken oberen Ecke des übergeordneten Objektes, bzw. bei Wurzelobjekten zum Bildschirm-Nullpunkt.
12	pb_y (Word): Pixelorientierte Y-Position des Objektes, relativ zur linken oberen Ecke des übergeordneten Objektes, bzw. bei Wurzelobjekten zum Bildschirm-Nullpunkt.
14	pb_w (Word): Pixelorientierte Breite des Objektes.
16	pb_h (Word): Pixelorientierte Höhe des Objektes.
18	pb_xc (Word): X-Position der linken oberen Ecke des aktuellen Clipping-Rechtecks (siehe CLIP).
20	pb_yc (Word): Y-Position der linken oberen Ecke des aktuellen Clipping-Rechtecks (siehe CLIP).
22	pb_wc (Word): Breite des aktuellen Clipping-Rechtecks.
24	pb_hc (Word): Höhe des aktuellen Clipping-Rechtecks.
26	pb_parm (Long): Beliebiger 32-Bit-Parameter für die User-Routine.

20. GEM-Adressen/-Felder

ADDRIN — AES-Adreß-Input-Block

Var% = LPEEK(ADDRIN + Offset) **(lesen)**
LPOKE ADDRIN + Offset,4-Byte-Adresse **(schreiben)**

Start-Adresse des AES-Adreß-Input-Blocks.

Für Version 3.0

ADDRIN() — Aus AES-ADDRIN-Block als Array lesen
ADDRIN()=
{ Ad)= } — In AES-ADDRIN-Block als Array schreiben

Var% = ADDRIN(Index) **(lesen)**
ADDRIN(Index) = 4-Byte-Adresse **(schreiben)**

4-Byte-Integerfeld. Entspricht AES-Addrin-Block (Index = Offset DIV 4).

ADDROUT — AES-Adreß-Output-Block

Var% = LPEEK(ADDROUT + Offset) **(lesen)**
LPOKE ADDROUT + Offset,4-Byte-Adresse **(schreiben)**

Start-Adresse des AES-Adreß-Output-Blocks.

Version 3.0

ADDROUT() — Aus AES-ADDROUT-Block als Array lesen
ADDROUT()=
{ Addro)= } — In AES-ADDROUT-Block als Array schreiben

Var% = ADDROUT(Index) **(lesen)**
ADDROUT(Index) = 4-Byte-Adresse **(schreiben)**

4-Byte-Integerfeld. Entspricht AES-Addrout-Block (Index = Offset DIV 4).

CONTRL

VDI - Kontrollblock

Var=DPEEK(CONTRL+Offset) **(lesen)**
DPOKE CONTRL+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des VDI-Kontroll-Blocks.

I/O:	Offset:	Bedeutung:
-->	+0	Funktions-Opcode
-->	+2	PTSIN-Array-Länge, geteilt durch 2
<--	+4	PTSOUT-Array-Länge, geteilt durch 2
-->	+6	INTIN-Array-Länge
<--	+8	INTOUT-Array-Länge
-->	+10	Evtl. Unterfunktions-Identifikator
<-->	+12	Gerätekennung (VDI-Handle)
<-->	+14 bis 28	Funktionsabhängige Informationen

Version 3.0

CONTRL()

Aus VDI-CONTRL-Block als Array lesen

CONTRL()= {CON)=}

In CONTRL-Block als Array schreiben

Var&=CONTRL(Index) **(lesen)**
CONTRL(Index)=2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. VDI-Contrl-Block (Index = Offset DIV 2).

GB

Startadresse des Array-Zeigerblocks

Var%=LPEEK(GB+Offset) **(lesen)**
LPOKE GB+Offset,4-Byte-Adresse **(schreiben)**

Start-Adresse des Pointer-Blocks für AES-Arrays.

Offset:	Bedeutung:
+0	Zeiger auf GCONTRL-Array-Start
+4	Zeiger auf Global-Array-Start
+8	Zeiger auf GINTIN-Array-Start
+12	Zeiger auf GINTOUT-Array-Start
+16	Zeiger auf ADDRIN-Array-Start
+20	Zeiger auf ADDROUT-Array-Start

GCONTRL — AES-Kontroll-Block

Var=DPEEK(GCONTRL+Offset) **(lesen)**
DPOKE GCONTRL+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des AES-Kontroll-Blocks.

```
I/O: Offset:  Bedeutung:

 -->  +0      Funktions-Opcode
 -->  +2      GINTIN-Array-Länge in Bytes
<--   +4      GINTOUT-Array-Länge in Bytes
 -->  +6      ADDRIN-Array-Länge (4 Bytes/Element)
<--   +8      ADDROUT-Array-Länge (4 Bytes/Element)
```

Version 3.0

GCONTRL() — Aus AES-CONTRL-Block als Array lesen
GCONTRL()= { GC)= } In CONTRL-Block als Array schreiben

Var&=GCONTRL(Index) **(lesen)**
GCONTRL(Index)=2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. AES-Contrl-Block (Index = Offset DIV 2).

GINTIN — AES - Integer-Inputblock

Var=DPEEK(GINTIN+Offset) **(lesen)**
DPOKE GINTIN+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des AES-Integer-Input-Blocks.

Version 3.0

GINTIN() — Aus AES-INTIN-Block als Array lesen
GINTIN()= { GI)= } In AES-INTIN-Block als Array schreiben

Var&=GINTIN(Index) **(lesen)**
GINTIN(Index)=2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. AES-Intin-Block (Index = Offset DIV 2).

GINTOUT — AES-Integer-Output-Block

Var=DPEEK(GINTOUT+Offset) **(lesen)**
DPOKE GINTOUT+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des AES-Integer-Output-Blocks.

Version 3.0

GINTOUT() — Aus AES-INTOUT-Block als Array lesen
GINTOUT()=

{ GINTO)= } — In INTOUT-Block als Array schreiben

Var&=GINTOUT(Index) **(lesen)**
GINTOUT(Index) = 2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspricht AES-Intout-Block (Index = Offset DIV 2).

INTIN — VDI-Integer-Input-Block

Var=DPEEK(INTIN+Offset) **(lesen)**
DPOKE INTIN+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des VDI-Integer-Input-Blocks.

Version 3.0

INTIN() — Aus VDI-INTIN-Block als Array lesen
INTIN()= { INT)= } — In VDI-INTIN-Block als Array schreiben

Var&=INTIN(Index) **(lesen)**
INTIN(Index)=2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. VDI-Intin-Block (Index = Offset DIV 2).

INTOUT — VDI-Integer-Output-Block

Var=DPEEK(INTOUT+Offset) **(lesen)**
DPOKE INTOUT+Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des VDI-Integer-Output-Blocks.

Version 3.0

INTOUT() — Aus VDI-INTOUT-Block als Array lesen
INTOUT()= { INTO)= } — In INTOUT-Block als Array schreiben

Var& = INTOUT(Index) **(lesen)**
INTOUT(Index) = 2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. VDI-Intout-Block (Index = Offset DIV 2).

PTSIN — VDI-Punkt-Input-Block

Var = DPEEK(PTSIN + Offset) **(lesen)**
DPOKE PTSIN + Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des VDI-Punkt-Input-Blocks.

Version 3.0

PTSIN() — Aus VDI-PTSIN-Block als Array lesen
PTSIN()= { PT)= } — In VDI-PTSIN-Block als Array schreiben

Var& = PTSIN(Index) **(lesen)**
PTSIN(Index) = 2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. VDI-Ptsin-Block (Index = Offset DIV 2).

PTSOUT — VDI-Punkt-Output-Block

Var = DPEEK(PTSOUT + Offset) **(lesen)**
DPOKE PTSOUT + Offset,2-Byte-Wert **(schreiben)**

Start-Adresse des VDI-Punkt-Output-Blocks.

Version 3.0

PTSOUT() — Aus VDI-PTSOUT-Block als Array lesen
PTSOUT()= { PTSO)= } — In PTSOUT-Block als Array schreiben

Var& = PTSOUT(Index) **(lesen)**
PTSOUT(Index) = 2-Byte-Wert **(schreiben)**

2-Byte-Integerfeld. Entspr. VDI-Ptsout-Block (Index = Offset DIV 2).

VDIBASE — VDI-Verwaltungsblock

Var=VDIBASE

Start-Adresse eines Speicherbereichs, den GEM zur Verwaltung der VDI-Parameter einrichtet. Diese Liste ist nicht für das MEGA- oder Blitter-TOS gültig. In V3.0 werden hier konfuse Werte geliefert. Beim Standard-ROM-TOS liegt jedoch der Parameterblock für die virtuelle Workstation ab Adresse &H578C. Dieser Block ist ebenfalls 308 Bytes lang und hat denselben Aufbau wie die GFA-VDIBASE. Die GFA-VDIBASE ist eine Kopie dieses Parameterblocks. In V3.0 können Sie also statt VDIBASE diese Blockadresse verwenden.

Offsets:

+0 (Word) Aktuelle Textrotation (0/900/1800/2700).
+2 (Word) Clipping-Flag (1 = An/0 = Aus, siehe CLIP).
+4 (Long) Adresse des aktuellen Font-Headers (8x16 oder 8x8).
+8 (Word) DDA_INC (?).
+10 (Word) Multifill-Flag.
+12 (Word) Anzahl der Scan-Lines des akt. Füllmusters (Füllmuster liegt ab LPEEK(VDIBASE+14)).
+14 (Long) Füllmusterdaten-Startadresse.
+18 (Word) Points/Absolute Mode (?).
+20 (Long) scrtchp (Puffer-Pointer/akt. Boot-Sektor ??).
+24 (Word) scrpt2 (Minus-Offset zu Lpeek(Vdibase+20) ??). Ergibt Start des evtl. ausführbaren Boot-Sektor-Teils -> ab Lpeek(Vdibase+20)-Dpeek(Vdiabse+24) liegen die letzten 490 Bytes des Boot-Sektors ??.
+26 (Word) Aktueller Textstil.
+28 (Word) T_sclsts (=0 ??).
+30 (Word) Aktuelle Polygon-Füllfarbe.
+32 (Word) Aktuelles Füllmuster (= Deffill Farbe,Dpeek(Vdibase+32)-1,Stil)
+34 (Word) Umrandungs-Flag (an/aus siehe BOUNDARY).
+36 (Word) Aktueller Füllstil.
+38 (Word) Aktuelle horizontale Textausrichtung.
+38 (Word) (0 = Links/1 = Zentriert/2 = Rechts).
+40 (Word) VDI-Handle der akt. Workstation (GFA-BASIC).
+42 (Word) Aktuelle Linienstartform.
+44 (Word) Aktuelle Linienfarbe.
+46 (Word) Aktuelle Linienendform.
+48 (Word) Aktueller Linienstil.
+50 (Word) Aktuelle Linienbreite.
+52 (Long) Zeiger auf nächsten VDI-(GDOS)-Font.
+56 (Word) Aktuelle Polymarkerfarbe.
+58 (Word) Aktuelle Polymarkerhöhe.
+60 (Word) Aktueller Polymarkertyp.
+62 (Word) Relative Polymarkerhöhe.
+64 (Long) Zeiger auf nächsten virt. Workstation-Parameter-Block.

+68	(Word) Anzahl der verfügbaren GEM/VDI-Fonts.
+70	(Word) Index des aktuellen GEM/VDI-Fonts.
+72 bis +161	(88 Bytes) Kopie des aktuellen Font-Headers (siehe unten).
+162	(Word) Aktuelle Textfarbe.
+164	(Word) Aktueller Benutzer-Linienstil.
+166 bis +197	(16 Words) Plane 1 des Benutzer-Füllmuster (Hires).
+198 bis +229	16 (Words) Plane 2 des Benutzer-Füllmusters (COLOR).
+230 bis +261	(16 Words) Plane 3 des Benutzer-Füllmusters (COLOR).
+262 bis +293	(16 Words) Plane 4 des Benutzer-Füllmusters (Hires).
+294	(Word) Aktuelle vertikale Textausrichtung: 0 = baseline 1 = halfline 2 = ascentline 3 = bottomline 4 = descentline 5 = topline
+296	(Word) Akt. Grafikmodus => Graphmode Dpeek(Vdibase+296)-1.
+298	(Word) Aktuelles Koordinatensystem: Normalized Device Coordinates (NDC) = 0 Raster Coordinates (RC) = 2 -> Standard
+300 bis +306	(4 Words) X1, Y1, X2, Y2 der akt. Clipping-Box (siehe CLIP ..TO..).

Aufbau eines Font-Headers

+0	(Word) Identifier (1 = System-Font).
+2	(Word) Größe des Fonts in Pixel
+4 bis +35	(32 Bytes) Font-Name als ASCII-String.
+36	(Word) ASCII des niedrigsten darstellbaren Zeichens.
+38	(Word) ASCII des höchsten darstellbaren Zeichens.

+40	(Word) Abstand der top line zur base line.
+42	(Word) Abstand der ascent line zur base line.
+44	(Word) Abstand der half line zur base line.
+46	(Word) Abstand der descent line zur base line.
+48	(Word) Abstand der bottom line zur base line.
+50	(Word) Bit-Breite des breitesten Zeichens.
+52	(Word) Bit-Breite der breitesten Zeichenbox.
+54	(Word) Kursiv-Toleranz links.
+56	(Word) Kursiv-Toleranz rechts.
+58	(Word) Fettschrift-Flag.
+60	(Word) Bei underlined-Text Pixel-Dicke des Strichs.
+62	(Word) Hellschrift-Maske (lightening grau = &H5555).
+64	(Word) Kursivschrift-Maske (normal skewing = &H5555).
+66	(Word) Multi-Flag. And 1 = 0 --> Kein System-Font And 1 = 1 --> System-Font And 2 = 0 --> Keine Horizontal-Offset-Tabelle And 2 = 1 --> Horizontal-Offset-Tabelle vorhanden And 4 = 0 --> Font im Intel-Format (HI/LO) And 4 = 1 --> Font im 68000er-Format (LO/HI) And 8 = 0 --> Variierende Zeichenbreite And 8 = 1 --> Alle Zeichen gleich breit
+68	(Long) Zeiger auf Horizontal-Offset-Tabelle.
+72	(Long) Zeiger auf Character-Offset-Tabelle.
+76	(Long) Zeiger auf die Font-Daten.
+80	(Word) Summe aller 256 Scan-Line-Breiten (--> Byte-Abstand der Scan-Lines im Puffer voneinander).
+82	(Word) Anzahl der Scan-Lines des Fonts (8x8 = 8/8x16 = 16).
+84	(Long) Zeiger auf nächsten verfügbaren Font.

Ab Offset +88 folgen dann 512 Bytes für die horizontale Offset-Tabelle, die für jedes Zeichen des Fonts separat den Bit-Abstand seiner ersten Scan-Line zum Font-Datenstart angibt.

21. Systemaufrufe

21.1 BIOS

BIOS() — BIOS-Routinen aufrufen

Var=BIOS(Opcode [,Parameterliste])

Opcode gibt die jeweilige BIOS-Funktionsnummer an. Parameterliste enthält die notwendigen Funktionsparameter. Sind Longwords zu übergeben, wird dem entsprechenden Wert ein "L:" vorangestellt. Wird die Angabe einer Kennung unterlassen, gelten die übergebenen Werte im Wordformat.

Nach Rückkehr kann der Inhalt von D0 (= Funktionsergebnis) ausgegeben (PRINT BIOS(...)), einer Variablen übergeben (A%=BIOS(...)), in Bedingungsabfragen (IF BIOS(...)) oder Ausdrücke (X$="BIOS-Rückgabe: "+STR$(BIOS(...))) eingebunden werden.

Var=BIOS(0,L:Adresse) *-GETMPB-*

Überträgt den GEMDOS-Memory-Parameter-Block in einen 12-Byte-Puffer ab Adresse. Dieser Block besteht aus drei Zeigern:

```
Adresse+0 (Longword) = Memory-Free-List          (mfl)
Adresse+4 (Longword) = Memory-Allocated-List     (mal)
Adresse+8 (Longword) = Roving-Pointer            (Umlaufzeiger)
```

Die beiden Memory..Lists sind folgendermaßen aufgebaut:

```
1. Longword = Zeiger auf Folgeblock         (link)
2. Longword = Zeiger auf Blockstart        (start)
3. Longword = Blocklänge                  (length)
4. Longword = Process-Descriptor             (own)
```

Ohne genaueste TOS-Kenntnisse gibt es hier keine Anwendungsmöglichkeiten.

Var=BIOS(1,Device) *-BCONSTAT-*

Liefert Auskunft, ob am angegebenen Device (Port) Bytes anliegen. Entspricht INP?(Device) (siehe dort).

Var=BIOS(2,Device) *-BCONIN-*

Wartet auf Dateneingang am Port Device. Kann ein Zeichen empfangen werden, wird ein 4-Byte-Wert zurückgegeben. Zu Device siehe INP(Device) und zum 4-Byte-Wert bei Device = 2, siehe KEYGET.

Var=BIOS(3,Device,Byte) *-BCONOUT-*

Gibt das angegebene Byte auf Port Device aus. Kehrt erst zurück, wenn Byte ausgegeben werden konnte (siehe OUT Device,Byte).

Var=BIOS(4,Flag,L:Adresse,Anzahl,Start,Laufwerk) *-RWABS-*

Liest/schreibt Anzahl Sektoren ab logischem Sektor Start von/auf Laufwerk in/aus Pufferbereich ab Adresse. Flag hat folgende Bedeutung:

```
0 = Sektoren lesen          (bei Disk-Wechsel Error )
1 = Sektoren schreiben      ( "       "         "  )
2 = Sektoren gelesen        (Disk-Wechsel ignorieren)
3 = Sektoren geschrieben    (    "           "     )
```

Start liegt im Bereich von 0 bis zur Gesamtsektorenanzahl-1. Für Anzahl gilt dasselbe.

Die Größe des Puffers muß 512 mal so groß sein, wie Sektoren gelesen, bzw. geschrieben werden sollen (1 Sektor = 512 Byte).

Laufwerk bestimmt das Laufwerk, auf welches sich die Funktion beziehen soll (0 = A/1 = B/2 = Hard-Disk). Ist Var = 0, so wurde fehlerfrei gelesen bzw. geschrieben.

Var=BIOS(5,Exc_nummer,L:Adresse) *-SETEXEC-*

Ändert den Exception-Vektor mit der angegebenen Exc_nummer auf Adresse. Ab Adresse muß die neue Exception-Routine liegen.

In Var wird die Adresse des alten Vektors geliefert. Soll nur die alte Adresse geliefert werden, ist in Exc_nummer eine -1 zu übergeben.

Var=BIOS(6) *-TICKCAL-*

Liefert Millisekunden-Differenz zwischen zwei Timer-Ticks (20).

Var=BIOS(7,Laufwerk) *-GETBPB-*

Liefert in Var die Adresse des BIOS-Parameterblocks für das angegebene Laufwerk (0 = A/1 = B). Dies ist - für jedes Laufwerk getrennnt - ein jeweils 18 Byte großer Puffer mit wichtigen Diskettendaten:

Word 1	Byte-Größe eines Sektors.
Word 2	Anzahl der Cluster pro Sektor.
Word 3	Byte-Größe eines Clusters.
Word 4	Directory-Länge in Sektoren.
Word 5	Größe eines FAT in Sektoren.
Word 6	Sektornummer des 2. FAT.
Word 7	Sektornummer des 1. Daten-Clusters.
Word 8	Anzahl der vorhandenen Daten-Cluster.
Word 9	Verschiedene Flags (Bedeutung unbekannt).

Var=BIOS(8,Device) *-BCOSTAT-*

Liefert in Var Auskunft, ob am angegebenen Device (Port) Bytes ausgegeben werden können. Entspricht OUT?(Device) (siehe dort).

Var=BIOS(9,Laufwerk) *-MEDIACH-*

Liefert in Var Auskunft, ob die Diskette im angegebenen Laufwerk (0 = A/1 = B) gewechselt wurde:

0	Diskette wurde sicher nicht gewechselt.
1	Diskette wurde möglicherweise gewechselt.
2	Diskette wurde sicher gewechselt.

Var=BIOS(10) *-DRVMAP-*

Liefert Information darüber, welche Laufwerke zur Zeit angeschlossen sind. In Var wird ein Bit-Vektor zurückgegeben, dessen Bits für die jeweiligen Laufwerke stehen (Bit 0 für A /Bit 1 für B/Bit 2 für C etc.). Das Sytem geht grundsätzlich davon aus, daß mindestens A und B angeschlossen sind. Auch wenn nur A vorhanden ist, wird trotzdem der Wert 3 geliefert.

Var=BIOS(11,Status) *-KBSHIFT-*

Liefert in Var den Status der Umschalt-Tasten, wenn in Status der Wert -1 übergeben wird.

Bit 0 gesetzt	Rechte <Shift>-Taste ist gedrückt.
Bit 1 gesetzt	Linke <Shift>-Taste ist gedrückt.
Bit 2 gesetzt	<Control>-Taste ist gedrückt.
Bit 3 gesetzt	<Alternate>-Taste ist gedrückt.
Bit 4 gesetzt	<CapsLock> ist angeschaltet.

Bit 5 gesetzt	<Alternate><ClrHome> ist gedrückt.
Bit 6 gesetzt	<Alternate><Insert> ist gedrückt.

Wird in Status ein positiver Wert übergeben, so wird dies als gewünschter Status nach obigem Schema interpretiert. Es können so die Tastendrücke simuliert werden.

Dabei gilt dieser Status bis erneut eine der Umschalttasten gedrückt oder ein neuer Status BIOS(11) zugewiesen wird.

21.2 GEMDOS

GEMDOS() — GEMDOS-Routinen aufrufen

Var=GEMDOS(Opcode [,Parameterliste])

Opcode gibt die jeweilige GEMDOS-Funktionsnummer an. Parameterliste enthält die notwendigen Funktionsparameter. Sind Longwords zu übergeben, wird dem entsprechenden Wert ein "L:" vorangesellt. Wird die Angabe einer Kennung unterlassen, gelten die übergebenen Werte im Word-Format.

Nach Rückkehr kann der Inhalt von D0 (=Funktionsergebnis) ausgegeben (PRINT GEMDOS(...)), einer Variablen übergeben (A%=GEMDOS(...)), in Bedingungsabfragen (IF GEMDOS(...)) oder Ausdrücke (X$="GEMDOS-Rückgabe: "+STR$(GEMDOS(...))) eingebunden werden.

Das GEMDOS ist die Hauptbibliothek zur Bereitstellung von grundlegenden Ein- und Ausgabefunktionen.

Bei einigen GEMDOS- und BIOS-Routinen wird bei fehlerhaft ausgeführter Funktion ein Fehler-Code zurückgegeben:

-32	Ungültige Funktionsnummer.
-33	Angegebene Datei nicht gefunden.
-34	Angegebener Pfadname existiert nicht.
-35	Zu viele offene Dateien.
-36	Datei-Zugriff ist nicht möglich.
-37	Ungültiges Datei-Handle.
-39	Speicher ist nicht ausreichend.
-40	Ungültige Speicherblock-Adresse.
-46	Ungültiges Laufwerk.
-49	Weitere Dateien sind nicht vorhanden.

Wird eine Null geliefert, heißt das, daß die Funktion korrekt ausgeführt wurde.

Var=GEMDOS(0) -*PTERM*-

Beendet das aktuelle Programm und kehrt zu Desktop bzw. zum Aufrufer zurück. In GFA-BASIC nicht verwenden, sondern QUIT bzw. SYSTEM einsetzen!

Var=GEMDOS(1) -*CONIN*-

Wartet wie INP(2) oder KEYGET auf die Eingabe eines Zeichens über die Tastatur. Es wird ein 32-Bit-Wert (siehe KEYGET) zurückgegeben und das entsprechende Zeichen an der aktuellen Cursor-Position ausgegeben.

Auf keinen Fall <Control><C> drücken, da Absturzgefahr!

Var=GEMDOS(2,Ascii) -*CONOUT*-

Gibt das durch den Byte-Wert Ascii angegebene Zeichen an der aktuellen Cursor-Position auf dem Bildschirm aus.

Var=GEMDOS(3) -*AUXIN*-

Wartet auf ein am Auxilliary-Port (RS232) eingehendes Zeichen und gibt dessen ASCII-Code ggfs. in Var zurück.

Var=GEMDOS(4,Ascii) -*AUXOUT*-

Wartet darauf, daß das Zeichen mit dem angegebenen ASCII-Wert am Auxilliary-Port (RS232) ausgegeben werden kann und gibt es ggfs. aus.

Var=GEMDOS(5,Ascii) -*PRNOUT*-

Wartet darauf, daß das Zeichen mit dem angegebenen ASCII-Wert am Centronics-Port (Drucker-Port) ausgegeben werden kann und gibt es ggfs. aus.

Var=GEMDOS(6,Ascii) *-RAWCONIO-*

Gibt das in Ascii angegebene Zeichen an der aktuellen Cursor-Position auf dem Bildschirm aus und kehrt zurück.

Wird in Ascii der Wert 255 übergeben, wird ein 32-Bit-Wert (siehe KEYGET) in Var geliefert, der dem zuletzt im Tastaturpuffer eingegangenen Zeichen entspricht (siehe INKEY$ bzw. KEYTEST). Es erfolgt dann keine Ausgabe.

Var=GEMDOS(7) *-RAWCONIN-*

Ist prinzipiell mit GEMDOS(1) identisch. Das gelesene Zeichen wird jedoch nicht gleichzeitig ausgegeben. <Control><C> kann hiermit gefahrlos gelesen werden.

Var=GEMDOS(8) *-RAWNECIN-*

Absolut identisch mit GEMDOS(7).

Var=GEMDOS(9,L:Adresse) *-CONWS-*

Gibt den ab Adresse liegenden String an der aktuellen Cursor-Position aus. Der String muß mit einem Null-Byte enden.

Var=GEMDOS(10,L:Adresse) *-CONRS-*

Liest einen String von der Tastatur ein (vgl. INPUT/INPUT$). In Adresse ist als Byte die maximale Länge der Eingabezeile vor Aufruf einzutragen. Daran anschließend ist ein Puffer in der benötigten Größe +1 vorzubereiten. Nach String-Eingabe wird vom GEMDOS in Adresse+1 (und 'Var') die tatsächlich eingegebene String-Länge und ab Adresse+2 der eingegebene String eingetragen.

Auf keinen Fall <Control><C> eingeben, da Absturzgefahr!

Tastatur-Kommandos während der Eingabe:

<Control><H>	Backspace (Cursor 1 Zeichen zurück)
<Control><I>	Tab (Cursor 8 Zeichen weiter)
<Control><J>	Eingabe-Ende und Line Feed ausgeben
<Control><M>	Eingabe-Ende und Carriage Return ausgeben
<Control><R>	Eingegebene Zeile in neuer Zeile ausgeben
<Control><U>	Eingabe ignorieren. In nächster Zeile neu.
<Control><X>	Eingabe löschen und Cursor an Zeilenanfang

Var=GEMDOS(11) *-CONSTAT-*
Liefert den Status des Tastaturpuffers (vgl. INP?(2) bzw. KEYLOOK).

```
    0   = Kein Zeichen verfügbar
65535 = Zeichen verfügbar
```

Var=GEMDOS(14,Laufwerk) *-SETDRV-*
Legt das angegebene Laufwerk (A = 0/B = 1/C = 2 etc.) als Default-Laufwerk fest. Entspricht CHDRIVE Laufwerk-1.

Im Gegensatz zu CHDRIVE liefert die Funktion in Var die Kennung des vorher aktiven Laufwerks.

Var=GEMDOS(16) *-CONOUTSTAT-*
Prüft Ausgabebereitschaft des Bildschirms. In GFA **nicht** verwenden!

Var=GEMDOS(17) *-PRTOUTSTAT-*
Liefert den Wert Null, wenn Ausgabe auf dem Drucker **nicht** möglich ist (vgl. OUT?(0)).

Var=GEMDOS(18) *-AUXINSTAT-*
Liefert den Wert &HFFFF, wenn am AUX-Port (RS232) ein Zeichen anliegt, sonst Null (vgl. INP?(1)).

Var=GEMDOS(19) *-AUXOUTSTAT-*
Liefert den Wert &HFFFF, wenn der AUX-Port (RS232) zur Ausgabe bereit ist, sonst Null (vgl. OUT?(1)).

Var=GEMDOS(25) *-CURDRV-*
Ermittelt das aktuelle Default-Laufwerk (0 = A/1 = B/2 = C etc.).

Var=GEMDOS(26,L:Adresse) *-SETDTA-*
Entspricht exakt der V3.0-Funktion FSETDTA() (siehe dort).

Var=GEMDOS(32,L:Adresse) *-SUPER-*

Schaltet in den Supervisor-Modus und zurück. Im Supervisor-Modus ist der Supervisor-Bereich unterhalb der Adresse 2048 auch durch Normal-POKEs und BMOVE etc. erreichbar.

Wird erstmalig in Adresse der Wert 0 übergeben, so wird der Superstack mit dem aktuellen Userstack-Inhalt geladen. In Var erhält man dann den letzten Inhalt des alten Superstacks (unbedingt merken!).

Das GEMDOS läuft jetzt im Supervisor-Modus (größte Vorsicht im Umgang mit POKEs etc.!). Dabei sind weiterhin auch GEMDOS-, BIOS- und XBIOS-Aufrufe möglich.

Wird erstmalig in Adresse ein Wert <>0 übergeben, so wird dies als gewünschter Superstack-Inhalt interpretiert und der Superstack damit geladen. Auch hier erhält man den alten Superstack-Inhalt in Var zurück. Das System bemerkt den Wechsel der Betriebsart und registriert GEMDOS(32)-Aufrufe abwechselnd als User->Super-Wechsel und Super->User-Wechsel. Der nächste Aufruf nach einem User->Super-Wechsel lädt den Superstack mit dem in Adresse übergebenen Wert (der beim ersten Aufruf erhaltene Superstack-Inhalt!) und schaltet das System wieder in den User-Betrieb. BMOVE mit einem Ziel unter 2048 ergibt wieder eine 2-Bomben-Fehlermeldung. Da bei dieser Funktion evtl. auch die Register A1 und D1 verändert werden, ist sie unter GFA-BASIC nur mit besonderer Vorsicht einzusetzen.

```
On break cont           ! Abbruch unterdrücken
A%=Gemdos(32,L:0)       ! Alten Superstack holen
Bmove 0,Xbios(2),32000  ! Dies ist nur im Supervisor-Modus
'                       ! möglich!
Print "SUPER-BMOVE aus Supervisor-Bereich = OKAY !"
Print "Supervisor-Stack ab Adresse: ",A%
B%=Gemdos(32,L:A%)      ! Alten Stand restaurieren
Pause 60
Cls
Print "USER-BMOVE aus Supervisor-Bereich = ERROR !"
Print "Supervisor-Stack (=USP) ab Adresse: ",B%
Bmove 0,Xbios(2),32000 ! Im User-Modus nicht möglich (Error)
```

Var=GEMDOS(42) *-GETDATE-*

Liefert das aktuelle System-Datum. Da das Format hier bitorientiert ist, ist dies wesentlich leichter mit DATE$ zu erfahren.

```
(Var AND &X1111111000000000)+1980 = Jahr
(Var AND &X111100000)             = Monat
(Var AND &X11110)                 = Tag
```

Var=GEMDOS(43,Format) *-SETDATE-*

Bestimmt das aktuelle System-Datum. Da das Format hier bit-orientiert ist, ist dies wesentlich leichter mit SETTIME zu erreichen.

```
Format (vgl. GEMDOS(42)): ((Jahr-1980)*2^9)+(Monat*2^5)+Tag
```

Var=GEMDOS(44) *-GETTIME-*

Liefert die aktuelle System-Uhrzeit. Da das Format hier bit-orientiert ist, ist dies wesentlich leichter mit TIME$ zu erfahren.

```
(Var AND &X1111100000000000) = Stunde
(Var AND &X11111100000)      = Minute
(Var AND &X11111)*2          = Sekunde
```

Var=GEMDOS(45,Format) *-SETTIME-*

Bestimmt die aktuelle System-Uhrzeit. Da das Format hier bit-orientiert ist, ist dies wesentlich leichter mit SETTIME zu erreichen.

```
Format (vgl. GEMDOS(44)): (Stunde*2^11)+(Minute*2^5)+(Sekunde/2)
```

Var=GEMDOS(47) *-GETDTA-*

Ist absolut identisch mit der V3.0-Funktion FGETDTA() (siehe dort).

Var=GEMDOS(48) *-GETVERS-*

Liefert die aktuelle GEMDOS-Versionsnummer.

Var=GEMDOS(49,L:Bytes,Wert) *-KEEP PROCESS-*

Beendet das aktuelle Programm und kehrt zum Desktop bzw. zum Aufrufer zurück. Dabei wird ab der Basepage des zu beendenden Programms ein Speicher von Bytes Länge reserviert. Das Programm bleibt also resident. Der reservierte Bereich wird vom GEMDOS nicht mehr vergeben. Außerdem kann an den Prozeß, zu dem zurückgekehrt wird, in Wert ein 16-Bit-Wert übergeben werden. In GFA-BASIC sollte diese Funktion nicht eingesetzt werden. Sie wird bei Accessories und residenten Auto-Ordner-Programmen verwendet.

Var=GEMDOS(54,L:Adresse,Laufwerk) *-GETDFREE-*

In Adresse ist die Startadresse eines 12-Byte-Puffers zu übergeben. In diesem Puffer werden von der Funktion vier Longwords abgelegt, die den Speicherzustand des in Laufwerk angegebenen Laufwerks (1 = A/2 = B/3 = C etc.) beschreiben:

```
Adresse+0  (Longword) =  Anzahl freier Cluster auf Disk
Adresse+4  (Longword) =  Gesamt-Cluster-Anzahl der Disk
           (Standard  => SingleSide = 351/DoubleSided = 711)
Adresse+8  (Longword) =  Sektorgröße in Bytes (Standard = 512)
Adresse+12 (Longword) =  Sektoren pro Cluster  (Standard = 2)
```

Die Anzahl freier Bytes auf Laufwerk ergibt sich aus:

```
Lpeek(Adresse) * Lpeek(Adresse+12) * Lpeek(Adresse+8)
(freie Cluster)* (Sek. pro Cluster)* (Bytes pro Sek.)
```

Die Gesamtgröße von Laufwerk ergibt sich aus:

```
Lpeek(Adresse+4) * Lpeek(Adresse+12) * Lpeek(Adresse+8)
(Gesamt-Cluster) * (Sek. pro Cluster)* (Bytes pro Sek.)
```

Var=GEMDOS(57,L:Adresse) *-MKDIR-*

Erzeugt einen Ordner auf der aktuellen Station (siehe MKDIR).

In Adresse wird die Startadresse des gewünschten Ordnernamens angegeben. Der String muß mit einem Null-Byte abgeschlossen sein (C-Konventionen).

Var=GEMDOS(58,L:Adresse) *-RMDIR-*

Löscht den Ordner mit dem ab Adresse liegenden Namen (Abschluß durch Null-Byte). Der Ordner muß leer sein (vgl. RMDIR)

Var=GEMDOS(59,L:Adresse) *-CHDIR-*

Wechselt das Default-Directory. Der Pfadname der neuen Default-Ebene muß dabei ab Adresse liegen und durch ein Null-Byte abgeschlossen sein (vgl. CHDIR).

Var=GEMDOS(60,L:Adresse,Attribut) *-CREATE-*

Erläuterung wäre hier zu aufwendig. Entspricht ungefähr OPEN "O".

Var=GEMDOS(61,L:Adresse,Modus) -*OPEN*-
Erläuterung wäre hier zu aufwendig. Entspricht ungefähr OPEN "I" bzw. OPEN "U".

Var=GEMDOS(62,Kanal) -*CLOSE*-
Erläuterung wäre hier zu aufwendig. Entspricht ungefähr CLOSE #'Kanal'.

Var=GEMDOS(63,Kanal,L:Länge,L:Adresse) -*READ*-
Liest Länge Bytes aus der Datei Kanal mit Ziel Adresse. Erläuterung wäre hier zu aufwendig. Entspricht ungefähr BGET # Kanal,Adresse,Länge bzw. INPUT # Kanal.

Var=GEMDOS(64,Kanal,L:Länge,L:Adresse) -*WRITE*-
Schreibt Länge Bytes ab Adresse in die Datei Kanal.

Erläuterung wäre hier zu aufwendig. Entspricht ungefähr BPUT # Kanal,Adresse,Länge bzw. PRINT # Kanal.

Var=GEMDOS(65,L:Adresse) -*UNLINK*-
Löscht die Datei mit dem ab Adresse liegenden Namen, der durch ein Null-Byte abgeschlossen sein muß (siehe KILL).

Var=GEMDOS(66,L:Bytes,Kanal,Modus) -*LSEEK*-
Versetzt den File-Pointer der Datei Kanal.

Modus 0	**Auf das Byte Bytes der Datei (vgl. SEEK #Kanal,+Bytes).**
Modus 1	**Um Bytes Bytes ab der aktuellen Position (vgl. RELSEEK #Kanal,+/- Bytes).**
Modus 2	**Auf das Byte Bytes - ab File-Ende gezählt (vgl. SEEK #Kanal,-Bytes).**

Weitere Erläuterungen wären zu aufwendig (siehe SEEK/RELSSEK).

Var=GEMDOS(67,L:Adresse,Modus,Attribut) -*CHMODE*-
Ermittelt oder ändert das Attribut einer Datei.

Adresse	**Startadresse des Dateinamens (Null-Byte-Abschluß), deren Attribut ermittelt oder verändert werden soll.**

Modus	Gibt an, ob das Attribut ermittelt (0) oder geändert (1) werden soll.
Attribut	Gibt Attribut an, das der Datei durch Modus = 1 gegeben werden soll. 0 = Normal (lesen/schreiben) 1 = Nur-Lese-Datei 2 = Versteckte (hidden) Datei 4 = System-Datei 8 = Disketten-Label 16 = Unter-Inhaltsverzeichnis Bei Modus = 0 ist das Attribut unwichtig, sollte aber trotzdem angegeben werden (siehe auch FGETDTA()).

Var=GEMDOS(69,Kanal) *-DUP-*

Ordnet einem Ausgabekanal die File-Nummer Kanal (1 - 5) zu. Erläuterung wäre hier zu aufwendig. Entspricht ungefähr OPEN "", #Kanal,"VID:" bzw. "CON:" bzw. "PRN:" bzw. "AUX:".

Var=GEMDOS(70,Kanal,Standard) *-FORCE-*

Ordnet dem Standard-Ausgabekanal die neue File-Nummer Kanal zu. Dies ist eine gegenüber GEMDOS(69) erweiterte Funktion.

Erläuterung wäre hier zu aufwendig. Entspricht ungefähr OPEN "",#Kanal,"VID:" bzw. "CON:" bzw. "PRN:" bzw. "AUX:".

Var=GEMDOS(71,L:Adresse,Laufwerk) *-GETDIR-*

Schreibt in den ab Adresse liegenden Puffer den Namen des aktuell geöffneten Ordners auf dem angegebenen Laufwerk (0 = Aktuell/1 = A/2 = B/3 = C etc.). Entspricht DIR$ (siehe dort).

Var=GEMDOS(72,L:Bytes) *-MALLOC-*

Entspricht exakt der V3.0-Funktion MALLOC() (siehe dort).

Var=GEMDOS(73,L:Adresse) *-MFREE-*

Entspricht exakt der V3.0-Funktion MFREE() (siehe dort).

Var=GEMDOS(74,0,L:Adresse,L:Bytes) *-SETBLOCK-*

Entspricht exakt der V3.0-Funktion MSHRINK() (siehe dort).

Var=GEMDOS(75,Modus,L:Progname,L:Command,L:Environ)-PEXEC-
Entspricht EXEC Modus,Progname$,Command$,Environ$. Hier werden jeweils die Startadressen der Strings bzw. der Basepage übergeben, wobei die Strings alle mit einem Null-Byte enden müssen. Modus siehe unter EXEC (bei Modus 4 ist Com hier numerisch).

Var=GEMDOS(76,Wert) *-PTERM-*
Entspricht grundsätzlich GEMDOS(49) KeepProcess. Es wird jedoch kein Speicher reserviert.

Var=GEMDOS(78,L:Name_adresse,Attribut) *-SFIRST-*
Entspricht exakt der V3.0-Funktion FSFIRST() (siehe dort).

Var=GEMDOS(79) *-SNEXT-*
Entspricht exakt der V3.0-Funktion FSNEXT() (siehe dort).

Var=GEMDOS(86,L:Alt_adresse,L:Neu_adresse,0) *-RENAME-*
Ändert die Datei mit dem ab Alt_adresse liegenden Namen in den ab Neu_adresse liegenden Namen. Beide Strings müssen mit einem Null-Byte enden. Entspricht NAME bzw. RENAME (siehe dort).

Var=GEMDOS(87,L:Adresse,Kanal,Modus) *-GSDTOF-*
Ändert oder ermittelt die Datei-Zeit bzw. -Datum des Files mit dem angegebenen GEMDOS-Kanal. Erläuterung wäre hier zu aufwendig. Hat Ähnlichkeit mit dem V3.0-Befehl TOUCH #Kanal.

21.3 GEMSYS

GEMSYS { GE } AES-Routinen aufrufen

GEMSYS [[() Opcode ()]]

Opcode enthält die Funktionsnummer der jeweils aufzurufenden AES-Routine. Sollen nur die AES-Arrays verändert und dieselbe Funktion wiederholt werden, kann die Angabe von Opcode entfallen.

In V3.0 sind sämtliche AES-Aufrufe als Funktionen implementiert. Soweit möglich, sind die entsprechenden V2.xx-Prozeduren unter der jeweiligen V3.0-Funktion aufgeführt.

21.4 VDISYS

VDISYS {V} — VDI-Routinen aufrufen

VDISYS [[() Opcode [)]]
VDISYS [[() Opcode [,I_len,P_len[,Id]] [)]] **(nur V3.0)**

Opcode enthält die Funktionsnummer der jeweils aufzurufenden VDI-Routine. Sollen nur die VDI-Arrays verändert und dieselbe Funktion wiederholt werden, kann die Angabe von Opcode entfallen. In CONTROL+12 ist bei VDI-Aufrufen das jeweilige Workstation-Handle zu übergeben. Dieses kann in V2.xx aus VDIBASE+40 ausgelesen oder - in V3.0 - durch V~H ermittelt werden.

In Version V3.0 kann zusätzlich zum Opcode in I_len und P_len die Länge des INTIN-Arrays und des PTSIN-Arrays übergeben werden (siehe dort). Bei manchen Funktionen ist die Angabe eines Unterfunktions-Identifikators notwendig. Dieser kann ggfs. in Id übergeben werden.

Die VDI-Bibliothek bietet einige sehr komfortable Routinen an, die es Ihnen erlauben, Ihre Programme vielfältiger auszustatten. Viele dieser Routinen sind vollwertig in BASIC-Befehle übernommen worden, die hier nicht aufgeführt zu werden müssen. Die folgenden vier Routinen bieten die Möglichkeit, verschiedene aktuelle Grafik-Einstellungen zu erfahren. Gerade für die Produktion von Utilities und Befehlserweiterungen sind diese Routinen interessant, da solche Hilfsprogramme die aktuellen Einstellungen sowenig wie möglich verändern bzw. diese nach Abschluß der Arbeiten wieder weitestgehend restaurieren sollten.

Für die V2.xx-Anwender wurde versucht, unter der Befehlsbeschreibung zu den in V3.0 neu implementierten VDI-Operationen eine entsprechende Prozedur anzubieten, die es dann ermöglichen soll, auch in V2.xx die V3.0-Vorzüge genießen zu können. Dies ist auch für diejenigen interessant, die in V3.0 arbeiten, aber außerdem noch V2.xx-Programme schreiben, um diese dann compilieren zu können. V2.xx-Programme mit diesen Erweiterungen lassen sich dann natürlich wesentlich leichter in die V3.0-Syntax übertragen, sobald der V3.0-Compiler verfügbar ist.

VDISYS 35 -CURRENT POLYLINE ATTRIBUTS-

```
V_h%=Dpeek(Vdibase+40) ! VDI-Handle für V2.xx
' V_h%=V~H             ! "      "   für V3.0
@Getline(V_h%,*Lcol%,*Ltyp%,*Lthic%,*Lsta%,*Lend%,*Gmod%)
Print "Aktuell: DEFLINE ";Ltyp%;",";Lthic%;
Print ",";Lsta%;",";Lend%
Print "           COLOR ";Lcol%
Print "           GRAPHMODE ";Gmod%
Procedure Getline(Handle%,L1%,L2%,L3%,L4%,L5%,L6%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 35
  *L2%=Dpeek(Intout)   ! Linientyp
  *L1%=Dpeek(Intout+2) ! Linienfarbe
  *L6%=Dpeek(Intout+4) ! Grafikmodus
  *L4%=Dpeek(Intout+6) ! Linienstartform
  *L5%=Dpeek(Intout+8) ! Linienendform
  *L3%=Dpeek(Ptsout)   ! Liniendicke
Return
```

VDISYS 36 -CURRENT POLYMARKER ATTRIBUTS-

```
V_h%=Dpeek(Vdibase+40) ! VDI-Handle für V2.xx
' V_h%=V~H             ! "      "   für V3.0
@Getmark(V_h%,*Mcol%,*Mtyp%,*Msiz%,*Gmod%)
Print "Aktuell: DEFMARK ";Mcol%;",";Mtyp%;",";Msiz%
Print "           GRAPHMODE ";Gmod%
Procedure Getmark(Handle%,M1%,M2%,M3%,M4%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 36
  *M1%=Dpeek(Intout)+1 ! Markerfarbe
  *M2%=Dpeek(Intout+2) ! Markertyp
  *M4%=Dpeek(Intout+4) ! Grafikmodus
  *M3%=Dpeek(Ptsout+2) ! Markergröße
Return
```

VDISYS 37 -CURRENT FILL AREA ATTRIBUTS-

```
V_h%=Dpeek(Vdibase+40) ! VDI-Handle für V2.xx
' V_h%=V~H             ! "      "   für V3.0
@Getfill(V_h%,*Fcol%,*Ftyp%,*Fpat%,*Gmod%,*Bound%)
Print "Aktuell: DEFFILL ";Fcol%;",";Ftyp%;",";Fpat%
Print "           GRAPHMODE ";Gmod%
Print "           BOUNDARY  ";Bound%
Procedure Getfill(Handle%,F1%,F2%,F3%,F4%,F5%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 37
  *F2%=Dpeek(Intout)   ! Fülltyp
  *F1%=Dpeek(Intout+2) ! Füllfarbe
  *F3%=Dpeek(Intout+4) ! Füllmuster
  *F4%=Dpeek(Intout+6) ! Grafikmodus
```

```
  *F5%=Dpeek(Intout+8) ! P-Umrahmung (Boundary)
Return
```

VDISYS 38 -CURRENT GRAPHIC TEXT ATTRIBUTS-

```
V_h%=Dpeek(Vdibase+40) ! VDI-Handle für V2.xx
' V_h%=V~H             !  "    "   für V3.0
@Gettext(V_h%,*Tcol%,*Tang%,*Twid%,*Thei%,*Tbwid%,*Tbhei%,*Gmod%)
Print "     Textfarbe       : ";Tcol%
Print "     Textwinkel      : ";Tang%
Print "     Zeichenbreite   : ";Twid%
Print "     Zeichenhöhe     : ";Thei%
Print "     Zeichenboxbreite: ";Tbwid%
Print "     Zeichenboxhöhe  : ";Tbhei%
Print "     GRAPHMODE ";Gmod%
Procedure Gettext(Handle%,T1%,T2%,T3%,T4%,T5%,T6%,T7%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 38
  '    Dpeek(Intout)       ! Verwendeter VDI-Font
  *T1%=Dpeek(Intout+2)     ! Textfarbe
  *T2%=Dpeek(Intout+4)     ! Rotationswinkel
  '    Dpeek(Intout+6)     ! Horiz. Textausrichtung
  '    Dpeek(Intout+8)     ! Vert. Textausrichtung
  *T7%=Dpeek(Intout+10)+1  ! Grafikmodus
  *T3%=Dpeek(Ptsout)       ! Zeichenbreite
  *T4%=Dpeek(Ptsout+2)     ! Zeichenhöhe
  *T5%=Dpeek(Ptsout+4)     ! Zeichenboxbreite
  *T6%=Dpeek(Ptsout+6)     ! Zeichenboxhöhe
Return
```

VDISYS 104 -SET FILL PERIMETER VISIBILITY-

```
@Boundary(0)             ! Nur für V2.xx (siehe BOUNDARY)
Deffill ,2,2
Pcircle 100,100,80
Procedure Boundary(Flag%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(1)+Mkl$(0)+Mki$(Dpeek(Vdibase+40))
  Bmove Varptr(Buff$),Contrl,14
  Dpoke Intin,Flag%
  Vdisys 104
Return
```

VDISYS 118 -EXCHANGE TIMER INTERRUPT VECTOR-

```
Procedure Vdi_timer(Adresse%,P1%)
  ' - Nur in V2.xx einsetzbar -
  ' Die ab Adresse liegende Maschinenroutine wird im
  ' VDI-Timer-Interrupt ausgeführt. In P1% wird der
  ' aktuelle Interrupt-Intervall in 1/1000 Sekunden
  ' zurückgegeben und in P2% die Adresse der vorher aktiven
  ' Routine.
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Dpeek(Vdibase+40))
```

```
    Buff$=Buff$+Mkl$(Adresse%)
    Bmove Varptr(Buff$),Contrl,18
    Vdisys 104
    *P1%=Dpeek(Intout)        ! Interrupt-Intervall in 1/1000 Sek.
    *P2%=Lpeek(Contrl+18)     ! Alte Routinen-Adresse
  Return
```

VDISYS 5 *-VDI ESCAPE FUNKTIONEN-*

Unter dieser Funktionsnummer sind insgesamt 20 Funktionen zusammengefaßt. Sie befassen sich überwiegend mit bildschirmorientierten Aufgaben.

Da die Einträge in die VDI-Arrays von der jeweiligen Unterfunktion abhängig sind, wird zuerst nur das einzige allgemein gültige Element belegt: das Handle.

```
Contrl(6)=V~h                     ! In V3.0
Dpoke Contrl+12,Dpeek(Vdibase+40) ! In V2.xx
```

In CONTRL+10 bzw. - in V3.0 - in CONTRL(5) wird je nach gewünschter Unterfunktion deren Identifikator Id eingetragen. Welche Elemente sonst noch zu belegen sind, hängt von der jeweiligen Unterfunktion ab. Nachdem diese Belegungen ebenfalls abgeschlossen sind, wird die Funktion mit VDISYS 5 ausgelöst. Anschließend können bei einigen Funktionen aus dem INTOUT-Array Rückgabewerte gelesen werden.

Id:

1	Zeichen-Raster des Bildschirms ermitteln. Standard: Hires u. Midres 80*25/Lowres 40*25 <- Zeilen=Dpeek(Intout) <- Spalten=Dpeek(Intout+2)
2	Von Pixel-Text auf Zeichen-Text umschalten.
3	Von Zeichen-Text auf Pixel-Text umschalten.
4	Cursor eine Zeile hoch.
5	Cursor eine Zeile runter.
6	Cursor ein Zeichen nach rechts.
7	Cursor ein Zeichen nach links.
8	Cursor in linke obere Ecke setzen (Home).
9	Bildschirm ab Cursor löschen.
10	Zeile ab Cursor löschen.
11	Cursor positionieren (siehe PRINT AT(Xp,Yp)). -> Dpoke Intin,Yp -> Dpoke Intin+2,Xp
12	String ab Cursor-Position ausgeben. Die einzelnen Zeichen des Strings müssen dabei der Reihe nach im INTIN-Array liegen: -> Dpoke Intin+1,Zeichen1 -> Dpoke Intin+3,Zeichen2

-> Dpoke Intin+5,Zeichen3
...
etc.
-> Dpoke Contrl+6,String_länge

13 Text invers schreiben (weiß auf schwarz).

14 Text normal schreiben (schwarz auf weiß).

15 Aktuelle Cursor-Position ermitteln (siehe CRSCOL/CRSLIN):
<- Crslin =Dpeek(Intout)
<- Crscol =Dpeek(Intout+2)

16 Nachfrage, ob Maus verfügbar:
<- Maus =Dpeek(Intout) ! (0 = Nein/1 = Ja)

17 Hardcopy ausführen (siehe HARDCOPY bzw. XBIOS(20)).

18 Maus positionieren (siehe SETMOUSE):
-> Dpoke Contrl+2,2 ! 2 PTSIN-Elemente
-> Dpoke Ptsin,Xpos
-> Dpoke Ptsin+2,Ypos

102 System-Font wählen (siehe Prozedur Sysfont unter ASC):
-> Dpoke Contrl+6,2 ! 2 INTIN-Elemente
-> Lpoke Intin,Fontadresse

21.5 XBIOS

XBIOS() — XBIOS-Routinen aufrufen

Var=XBIOS(Opcode [,Parameterliste])

Nach der Rückkehr kann der Inhalt von D0 (= Funktionsergebnis) direkt ausgegeben (PRINT XBIOS(...)), einer Variablen übergeben (A%=XBIOS(...)), in Bedingungsabfragen (IF XBIOS(...)) oder auch Ausdrücke (X$="XBIOS-Rückgabe: "+STR$(XBIOS(...))) eingebunden werden.

Das XBIOS ist eine erweiterte Bibliothek zur allgemeinen Verwaltung von Ein- und Ausgaben jeglicher Art. Kann eine Funktion nicht ordnungsgemäß ausgeführt werden, so wird als Rückgabewert ein Fehler-Code geliefert.

XBIOS-Fehlermeldungen:

Code	Meldung
-1	Allgemeiner Fehler.
-2	Station nicht empfangsbereit.
-3	Unbekannter Befehl.
-5	Ungültiger Befehl.
-6	Track nicht gefunden.
-7	Boot-Sektor nicht gültig.
-8	Sektor nicht gefunden.
-10	Schreibfehler.
-12	Allgemeiner Fehler.
-13	Floppy ist schreibgeschützt.

-14	Floppy wurde gewechselt.
-15	Gerät unbekannt.
-16	Fehlerhafter Sektor.
-17	Floppy nicht eingelegt.

Wird eine Null geliefert, heißt das, daß die Funktion korrekt ausgeführt wurde.

Var=XBIOS(0,Typ,L:Parameter_adresse,L:Prog_adresse) *-INITMOUSE-*

Es kann eine eigene Maschinenroutine angegeben werden, die die Mausverwaltung übernimmt. Es sind mehrere Parameter über einen Parameterblock zu übergeben. Die Parameterstrukturen und ihre Bedeutungen sind für mich (und bestimmt nicht nur für mich) absolutes Niemandsland, weshalb ich hier keine weiteren Auskünfte geben kann.

Var=XBIOS(1,Bytes) *-SSBRK-*

Von einem ROM-Modul aus kann hiermit vor Installierung des Betriebssystems direkt das Boot-ROM angesprochen werden, um am oberen Ende des Arbeitsspeichers die angegebene Anzahl an Bytes zu reservieren.

Var=XBIOS(2) *-PHYSBASE-*

Liefert die Anfangsadresse des physikalischen Bildschirmspeichers. Als physikalischer Bildschirm wird der Speicherbereich bezeichnet, der vom Video-Shifter ständig auf den sichtbaren Bildschirm übertragen wird.

Var=XBIOS(3) *-LOGBASE-*

Im Gegensatz zur physikalischen Screen existiert ein logischer Bildschirm. Diese ist im allgemeinen mit der physikalischen Screen identisch, solange nicht durch XBIOS(5) unterschiedliche Startadressen zugewiesen werden.

Alle Bildschirmausgaben beziehen sich auf diesen logischen Bildschirm. D.h., daß von den Ausgaben nichts zu sehen ist, wenn Physbase und Logbase mehr als 32 KByte auseinanderliegen.

Var=XBIOS(4) *-GETREZ-*

Liefert die aktuelle Auflösung des Bildschirms.

```
0 = 320 * 200 Punkte (Lowres = 16 von 512 Farben)
1 = 640 * 200 Punkte (Midres =  4 von 512 Farben)
2 = 640 * 400 Punkte (Hires  =  schwarz / weiß)
```

Var=XBIOS(5,L:Logbase,L:Physbase,Resolution) *-SETSCREEN-*

Der logische und physikalische Bildschirm kann an verschiedene Speicheradressen gelegt werden:

Logbase Neue Startadresse der logischen Screen.
Physbase Neue Startadresse der physikalischen Screen.
Resolution
Neue Bildschirm-Auflösung. Der dritte Parameter ist nicht im GEM-Betrieb nutzbar, also in GFA-BASIC uninteressant. Hier ist generell der Wert -1 zu übergeben.

Die beiden "Bildschirme" **müssen** an einer durch 256 teilbaren Adresse beginnen. Ist dies nicht der Fall, kommt es fast immer zu Abstürzen. Den Einsatz von XBIOS(5) können Sie in den BMOVE-Beispielen beobachten (Prozedur Screen). Diese Funktion bildet das A und O zur Erzeugung flimmerfreier Animation.

Var=XBIOS(6,L:Adresse) *-SETPALETTE-*

Überträgt die ab Adresse liegenden 16 Words (32 Bytes) "en bloc" in die Farbregister. Das erste Word liegt dann in Register 0, das zweite in Register 1 usw.

Ein Word entspricht dem unter SETCOLOR beschrieben Gesamtfarbwert. Bei Degas-Bildern liegt z.B. die komplette Farbpalette des Bildes in den Bytes 3 - 34 der Bilddatei. Byte 1 und 2 (1. Word) enthalten die Screen-Auflösung, unter welcher das Bild abgespeichert wurde (entspricht XBIOS(4)). Die letzten 32000 Bytes der Bilddatei enthalten die Grafikdaten.

Var=XBIOS(7,Register,Farbwert) *-SETCOLOR-*

Belegt das angegebene Farb-Register mit dem angegebenen Farbwert (entspricht SETCOLOR Register,Farbwert).

Ist Farbwert = -1, so wird in Var der aktuelle Farbwert des angegebenen Registers zurückgegeben. Das Register bleibt dann unverändert.

Var=XBIOS(8,L:Adresse,L:0,Laufwerk,Sektor,Track,Side,Anzahl)
-FLOPREAD-

Liest Sektoren von einer Diskette in einen vorbereiteten Pufferbereich.

Adresse	Startadresse des Puffers. Der Puffer muß mindestens 512*Anzahl Bytes groß sein.
Laufwerk	Nummer des Laufwerks (A = 0/B = 1).
Sektor	Nummer des track-relativen Startsektors: (Standard: 1 - 9 -> *ein* Track = 9 Sektoren)
Track	Nummer des Tracks (Standard: 0 - 79).
Side	Disk-Seite (Single = 0/Double = 0 oder 1).
Anzahl	Anzahl der zu lesenden Sektoren: (Standard: max. 9-Sektor+1)

Var=XBIOS(9,L:Adresse,L:0,Laufwerk,Sektor,Track,Side,Anzahl)
-FLOPWRITE-

Schreibt Sektoren aus einem Speicherbereich auf eine Diskette.

Die Parameter sind mit den bei XBIOS(8) (Flopread) vergleichbar. Nur sinngemäß umgekehrt (z.B. schreiben statt lesen).

Var=XBIOS(10,L:Adr,L:0,Laufw,Anz,Trck,Sid,Intrl,L:Magc,Virgn)
-FLOPFMT-

Formatiert einzelne Tracks.

Adr	Startadresse eines 8000-Byte-Arbeitspuffers, in dem von der Funktion die Track-Daten erzeugt werden.
Laufw	Laufwerk (A = 0/B = 1).
Anz	Anzahl der Sektoren, in die der Track eingeteilt werden soll (Standard: 9). Es sind auch 10 Sektoren pro Track möglich (sog. Fat-Disks).
Trck	Nummer des Ziel-Tracks (Standard: 0 - 79). Bei Fat-Disks auch bis 81 möglich.
Sid	Disk-Seite (Single = 0/Double = 0 oder 1).
Interl	Gibt die Reihenfolge an, in welcher die Sektoren geschrieben werden sollen (!?) (Standard: 1).
Magc	Longword, das verseh. Formatieren erschwert (muß &H87654321 sein).
Virgn	Word-Muster, mit dem die Sektoren gefüllt werden sollen (Standard: &HE5E5 oder 0. Kann jeder beliebige Wert sein, die oberen 4 Bits der beiden Bytes dürfen jedoch nicht größer als 14 werden.

Das System gibt bei Überschreiten der maximalen Track-Anzahl keine Fehlermeldung aus (Vorsicht!). Bei evtl. auftretenden Fehlern wird in Var eine Fehlernummer zurückgeliefert. Konnten beim Formatieren Sektoren nicht formatiert werden, so erhält man -16 (bad sectors). In diesem Fall steht ab Pufferanfang eine mit einem Null-Byte abgeschlossene Word-Liste der fehlerhaften Sektor-Nummern.

Die Formatierung wird üblicherweise durch XBIOS(18) abgeschlossen.

Var=XBIOS(11) *Kein Effekt!*

Var=XBIOS(12,Bytes,L:Adresse) *-MIDIWS-*

Es werden Bytes+1 Bytes ab der angegebenen Adresse gelesen und als Einheit auf dem Midi-Out-Port ausgegeben.

Einfacher durch:

```
Open "O",#1,"MID:"
Print #1,"z.B. MIDI-Write-String....."
```

Var=XBIOS(13,Mfp_index,L:Adresse) *-MFPINT-*

Überschreibt im MFP-Interruptvektor den Sprungzeiger mit der Nummer Mfp_index durch die angegebene Adresse. Dadurch wird die ab Adresse liegende Maschinen-Routine im Interrupt ausgeführt.

Var=XBIOS(14,Device) *-IOREC-*

Liefert in Var die Startadresse des angegebenen I/O-Puffer-Datenblocks.

Device:

0	RS232 (Auxilliary)
1	Console (Tastatur)
2	MIDI

Aufbau der Datenblöcke:

1 Longword	
	Zeiger auf den jeweiligen Puffer.
1 Word	Größe des Puffers.
1 Word	Head. Zeiger auf die nächste Position hinter dem zuletzt eingegangenen Zeichen.
1 Word	Tail. Zeiger auf das als nächstes auszugebende Zeichen. Head und Tail sind Umlaufzeiger (Ringzeiger oder "rover"). Wird das oberste Puffer-Byte überschritten, springen Sie wieder auf den Pufferanfang. Ist Head = Tail, so ist der Puffer ohne Inhalt.
1 Word	Low water mark.
1 Word	High water mark.
	Arbeitet die RS232-Schnittstelle im Handshake-Modus (XON/XOFF bzw. RTS/CTS siehe XBIOS(15)) und im Eingangspuffer liegen mehr Bytes, als in *High water mark angegeben, so erhält der Sender* das Signal XOFF bzw. CTS. Weitere Eingänge werden dann solange abgewiesen, bis der Pufferinhalt auf die Low water mark abgesunken ist.

Für den RS232-Puffer wird durch XBIOS(14,0) nur die Adresse des Eingabe-Blocks geliefert. Der Block für den RS232-Ausgabepuffer liegt direkt hinter dem Block für den Eingabepuffer (AUX_Outputblock=XBIOS(14,0)+14).

Var=XBIOS(15,Baud,Modus,Usart,Rsr,Csr,Scr) *-RSCONF-*
Bietet die Möglichkeit, die RS232-Schnittstelle einzustellen.

Baud (Bits pro Sek.):

0 = 19200	1 = 9600	2 = 4800	3 = 3600
4 = 2400	5 = 2000	6 = 1800	7 = 1200
8 = 600	9 = 300	10 = 200	11 = 150
12 = 134	13 = 110	14 = 75	15 = 50

Modus (Kontrollparameter):

0	Handshake off (Default).
1	XON/XOFF.
2	RTS/CTS (RequestToSend/CloseToSend).
3	XON/XOFF und RTS/CTS gleichzeitig.

Usart (Usart-Register).

```
Bit 1 und 2  = no parity     &X0....000
             = odd parity    &X0....100
             = even .i.parity;  &X0....110
Bit 3 und 4  = kein Stopbit  &X0..00..0
             = 1 .i.Stopbit;    &X0..01..0
             = 1.5 Stopbits  &X0..10..0
             = 2 Stopbits    &X0..11..0
Bit 5 und 6  = 8 .i.Databit;s   &X000....0
             = 7 Databits    &X001....0
             = 6 Databits    &X010....0
             = 5 Databits    &X011....0
```

Rsr	MFP-ReceiverStatusRegister.
Tsr	MFP-TransmitterStatusRegister.
Scr	MFP-SynchronousCharacterRegister.

Wenn ein Parameter unverändert beibehalten werden soll, ist -1 zu übergeben.

Var=XBIOS(16,L:Normal,L:Shifted,L:Capslocked)-KEYTABLE-
Die Tastaturbelegung kann nach eigenem Bedarf organisiert werden.

Es existieren drei Tabellen, die die ASCII-Codes der Tasten bei drei verschiedenen Zuständen beinhalten. Der erste Zustand ist, wenn "normal" geschrieben wird, also weder <CapsLock> noch <Shift> gedrückt ist. Der zweite tritt ein, wenn eine <Shift>-Taste gedrückt wird und der dritte, wenn die <CapsLock>-Taste gedrückt wurde.

Die Tabellen sind je 128 Bytes lang. Ihre Adressen können durch diese Funktion ermittelt und ggfs. auch bestimmt werden. Jede Tabelle enthält der Reihe nach die ASCII-Codes der nach Scan-Code sortierten Tasten. Die Taste <F1> hat z.B. den Scan-Code 59. Soll bei Druck auf <F1> und gleichzeitig gedrückter <Shift>-Taste das Zeichen A ausgegeben werden, so muß in das 59. Byte der Shifted-Tabelle der Wert 65 (ASCII für A) eingetragen werden.

Von der Funktion wird in Var nach Abschluß ein Zeiger auf einen 12-Byte-Block (3 Longwords) geliefert, der nacheinander die drei Tabellen-Adressen enthält. Wird ein Parameter mit -1 angegeben, so bleibt die entsprechende Adresse im Zeigerblock unverändert. XBIOS(16,L:-1,L:-1,L:-1) liefert nur die Blockadresse, ohne die Einträge zu verändern.

```
On break cont             ! Break unterdrücken
Adr%=@Keyvec(-1,-1,-1)    ! Adr% enthält nun die Adresse der
'                         ! Normal-Tabelle. Adr%+4 die der
'                         ! Shifted- und Adr%+8 die der
'                         ! CapsLocked-Tabelle.
A$=Space$(128)            ! 128-Byte-Puffer vorbereiten
Bmove Lpeek(Adr%),Varptr(A$),128! Alte Tabelle im Puffer sichern
B$=Chr$(242)+Chr$(243)+Chr$(244)+Chr$(245)+Chr$(246)
B$=B$+Chr$(247)+Chr$(251)+Chr$(252)+Chr$(253)+Chr$(254)
'                         ! 1. String (F1-F10) mit ASCIIs
'                         ! belegen
C$=Chr$(23)+Chr$(24)+Chr$(25)+Chr$(20)+Chr$(21)
C$=C$+Chr$(22)+Chr$(17)+Chr$(18)+Chr$(19)+Chr$(16)
'                         ! 2. String (Ziffernblock-Zifferntasten)
'                         ! mit ASCIIs belegen
For I%=17 To 25           ! 3. String (Ziffernleiste-Zifferntaste)
  D$=D$+Chr$(I%)          ! mit gewünschten ASCIIs belegen
Next I%
D$=D$+Chr$(16)            ! ASCII der höchsten Taste (0) ist
'                         ! niedriger als der ASCII der untersten
'                         ! Zifferntaste (1).
Mid$(A$,60,10)=B$         ! String 1 in Puffer einsetzen
Mid$(A$,104,10)=C$        ! String 2 in Puffer einsetzen
Mid$(A$,3,10)=D$          ! String 3 in Puffer einsetzen
Void Xbios(16,L:Varptr(A$),L:-1,L:-1) ! Neue Tabellen-Adresse
'                                     ! übergeben
Print "F1-F10 oder Zifferntaste drücken"
Print "     (andere Taste=Abbruch)"
Repeat
  Key%=Inp(2)
  Out 5,Key%
```

```
Until Key%>25 And Key%<242
Void Xbios(24)          ! Alte Tabellen restaurieren
```

Um bei riskanten Änderungsversuchen hinterher die Tastatur noch benutzen zu können, wird hier das Break unterbunden, damit auf jeden Fall XBIOS(24) ausgeführt wird. Dadurch wird der ursprüngliche Zustand wieder hergestellt und die Tastatur kann wieder normal verwendet werden.

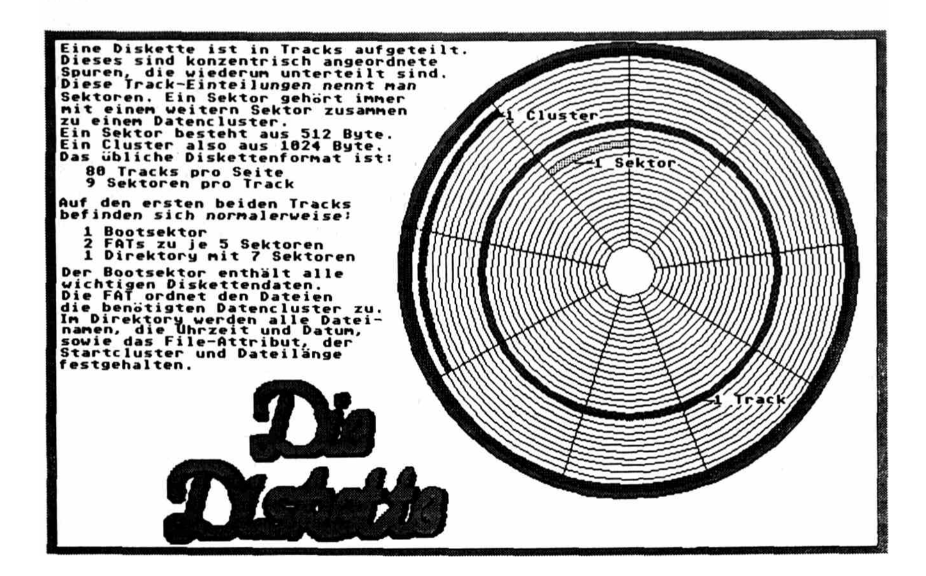
Var=XBIOS(17) *-RANDOM-*

Liefert eine 24-Bit-Zufallszahl (vgl. RANDOM()).

Diese Funktion habe ich so in eine DEFFN-Funktion abgeändert, daß eine Bit-Zahl angegeben werden kann. Wird z.B. als Bit-Zahl 4 übergeben, ist die höchste zurückgelieferte Zufallszahl eine &X1111 = 15. Die höchste verwendbare Bit-Zahl ist dabei 24.

```
Do
  Print Bin$(@Zufall(4))  ! 4 = 4 Bit-Zufallszahl
Loop
Deffn Zufall(Lim%)=Xbios(17) And ((2^(Lim% Mod 25))-1)
```

21.5.1 Aufbau einer Diskette



Inhalt und Aufbau der alles entscheidenden Disketten ist wohl den meisten Lesern ein Buch mit den berühmten sieben Siegeln. "Alles entscheidend" deshalb, weil unser ST ohne diese kleinen "Schwabbel-Scheiben" (engl.: Floppy-Disk) nun mal nicht auskommt.

So eine Diskette sieht eigentlich nicht besonders exklusiv aus. Es handelt sich um dünne Kunststoffscheiben, die mit einer magnetischen Oberfläche beschichtet sind. Die Daten werden grundsätzlich in der gleichen Verfahrensweise in diese magnetische Schicht eingraviert, wie wir es von Magnet-Tonbändern her kennen. Der Unterschied ist der, daß bei einem Tonband die Informationen analog zu der dem Tonkopf zugeführten Spannung übertragen werden, d.h. daß hier je nach Qualität des Gerätes und des Bandes eine sehr große Frequenzspanne vorliegen kann.

Bei einer Diskette werden ausschließlich digitale Informationen verarbeitet. Der "Tonkopf" überträgt also nur zwei Frequenzen. Die zu übertragenden Daten werden bitweise interpretiert und je nachdem, ob ein Bit gesetzt oder leer ist, entweder ein hoher oder ein niedriger Ton gesendet. Die Frequenz dieser beiden unterschiedlichen Töne hinterläßt dann auf der Magnetschicht ganz charakteristische Spuren.

Da sich die Diskette während des Schreibens und Lesens ständig um ihre Achse dreht, ist der Gedanke nicht fern, sie in konzentrische Kreise Kreise einzuteilen. Womit wir auch schon bei den mysteriösen Tracks (engl.: Spuren) angekommen sind. Sind Sie im Besitz eines besseren Formatier-Programms, wird Ihnen schon aufgefallen sein, daß die Anzahl an Tracks, die sich auf einer Diskette befinden, auch bei gleichem Diskettentyp unterschiedlich sein kann. Man hat dort also die Wahl, ob man die Diskette in 40, 80, 81 oder 82 Tracks (je Diskettenseite) einteilen möchte. Das bedeutet nichts anderes, als daß nun der zur Verfügung stehende Teil des Diskettenradius' physikalisch in die gewünschte Track-Anzahl unterteilt wird.

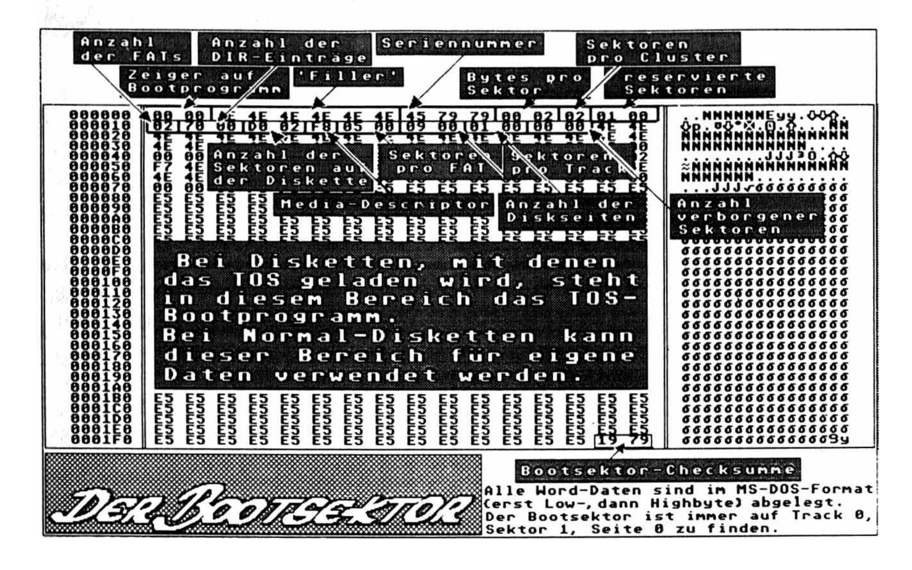
Weiterhin wird Ihnen vielleicht aufgefallen sein, daß auch die Anzahl der Sektoren variabel sein kann. Wird z.B. eine einseitige Diskette mit 82 Tracks zu je 10 Sektoren formatiert, stehen Ihnen 82*10 = 820 Sektoren zu je 512 Byte (820*512 = 419840 Byte) zur Verfügung. Da man auf diese Weise ca. 60 Kilobyte pro Diskettenseite mehr an Speicherplatz erhält, wird dieses Diskettenformat häufig verwendet.

Aus einem ganz bestimmten Grund erweist es sich jedoch oft als nachteilig, seine Diskette derart platzsparend einzurichten. Das Betriebssystem des ST ist in seinen Kopier- und Formatier-Utilities auf

das großzügigere "720-Sektoren-pro-Seite"-Format ausgerichtet und man hat so keine Möglichkeit, ohne spezielle Kopierprogramme seine Disketten zu vervielfältigen. Sie werden also üblicherweise Disketten vorfinden, die mit 80 Tracks zu je 9 Sektoren (80*9 = 720 Sektoren) ausgestattet sind.

Wer nun schnell mitgerechnet hat, kommt bei einseitigen Disketten auf eine Byte-Zahl von 720*512 = 368640 Bytes. Nach dem Formatieren mit dem Desktop-Formatter werden jedoch nur 357376 als frei gemeldet. Das liegt daran, daß das System sich einen "hübschen Batzen" von 22 Sektoren zur Diskettenverwaltung, also für Boot-Sektor, File-Allocation-Tables (engl.: Datei-Zuteilungs-Tabelle) und Directories abkneift. Es bleiben also nur noch 698 Sektoren (698*512 = 357376 Bytes) übrig. Ein erfreulicherweise für die meisten Zwecke völlig ausreichender Speicherplatz. Bei zweiseitigen Disketten steht die zweite Seite vollständig zur Datenspeicherung bereit, da diese 22 Sektoren nur von der ersten Seite abgezogen werden. Nach erfolgter Standard-Formatierung einer zweiseitigen Diskette wird Ihnen ein freier Speicherplatz von 726016 Bytes gemeldet ((698+720)*512 = 726016 Bytes).

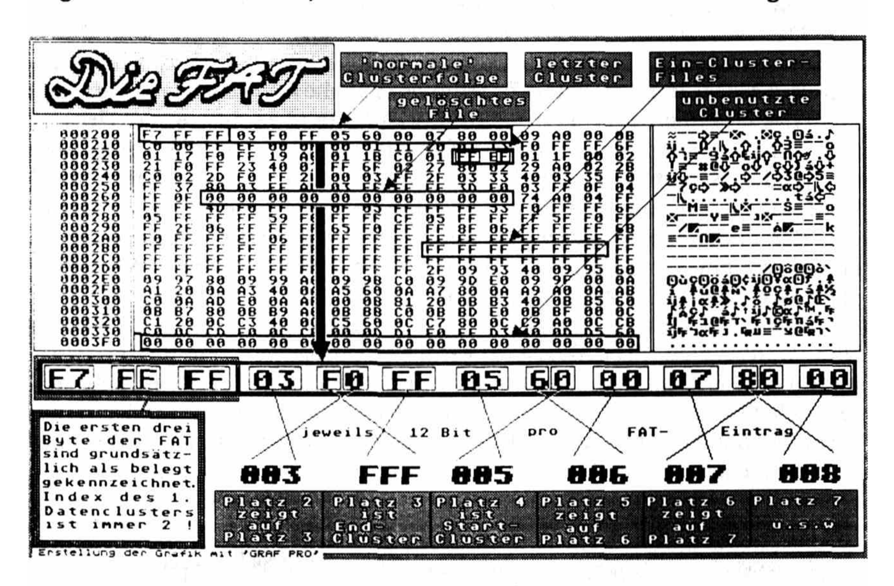
Nun sind gleich drei Begriffe gefallen, deren Erklärung zum Verständnis der Disketten-Organisation unumgänglich ist. Was sind eigentlich Sektoren? Sektoren sind die kleinste Einteilungsgröße auf der Diskette. Sie bestehen generell aus 512 Byte. Während ein Track eher eine technische Einheit ist, ist der Sektor zur Verwaltung der Diskette notwendig.



Boot-Sektor

Jede Diskette verfügt über einen solchen. Er ist generell auf Sektor 0 (Seite 0, Track 0) einer Diskette zu finden. Nach erfolgter Formatierung einer Diskette werden hier grundlegende Informationen zur Disketten-Organisation festgeschrieben (siehe unter XBIOS(18)).

Bei genauester Kenntnis der Materie ist es auch möglich, ihn nachträglich zu modifizieren, um so z.B. versteckte Tracks zu erzeugen.



File-Allocation-Table

Wie der Name schon sagt, wird hier den Dateien etwas zugewiesen. Wird eine Datei eingerichtet, werden in den FAT-Sektoren (üblich 2*5) die Sektoren eingetragen, die von dieser Datei belegt werden.

Eine FAT muß immer so groß sein, daß für jeden Sektor der Diskette eineinhalb Byte (12 Bit) zur Verfügung stehen. Üblicherweise werden pro Diskette zwei FATs angelegt, wovon die zweite direkt auf die erste folgt. Wurde z.B. eine einseitige Disk mit 80 Tracks zu je 9 Sektoren formatiert, muß eine FAT mindestens INT(((80*9)*1.5)/512)+1 = 3 Sektoren groß sein. Standardmäßig wird eine FAT grundsätzlich für eine doppelseitige Diskette ausgelegt (INT(((80*2*9)*1.5)/512)+1 = 5), woraus sich die übliche FAT-Größe von 5 Sektoren ergibt.

Da die erste FAT üblicherweise auf Sektor1 in Track 0 der Seite 0 einer Diskette direkt hinter dem Boot-Sektor liegt, folgt die zweite in diesem Fall also auf Sektor 6. Die FAT beginnt immer mit der 3-Byte-Konstante &HF7FFFF. Was jetzt folgt, ist nichts für schwache Nerven. Hier handelt es sich um einen echten Disketten-Dschungel.

Da sich durch ständiges Erstellen neuer und das Löschen alter Dateien die Cluster-Belegung auf der Diskette wie Spaghetti ineinander verknäult, haben sich die TOS-Bauer hier etwas ganz besonderes einfallen lassen.

Um nicht für jede neue Datei, die angelegt wird, neuen Speicherplatz zu verschwenden, wird in der FAT jeder Cluster, der nicht bzw. nicht mehr benötigt wird, mit einem 12-Bit-Wert von Null gekennzeichnet.

Wird nun eine Datei angelegt, geht das System die gesamte FAT durch und sucht der Reihe nach nach unbelegten Cluster-Plätzen. Dies macht es solange, bis die Anzahl der gefundenen Cluster (1 Cluster = 1024 Byte) ausreicht, um die Datenmenge der neuen Datei unterzubringen. Die Nummer des ersten gefundenen freien Clusters wird dann in der Direktory in das 27. und 28. Byte des neuen Direktory-Eintrags geschrieben.

Cluster — Ein Cluster besteht immer aus zwei Sektoren und der erste Daten-Cluster liegt direkt hinter den Direktory-Sektoren (üblicherweise ab Sektor 18). Die Cluster-Zählung beginnt bei Sektor 18 mit dem Index 2 (Sektor 18 und 19 = Cluster 2, Sektor 20 und 21 = Cluster 3 usw.).

Da wir aber konfuserweise nicht im ST-Format arbeiten, sondern im MS-DOS-Format, steht im 27. Byte das LO-Byte und im 28. Byte das HI-Byte dieser Zahl. Um die Nummer des ersten belegten Clusters dieser Datei zu erfahren, muß man also diesen Wert aus dem Direktory-Eintrag der betreffenden Datei auslesen.

Die durch die Datei belegten Cluster ermittelt man nun, indem man mit dieser Clusternummer nun in die FAT geht. Und zwar an den FAT-Platz, der für diesen betreffenden Cluster reserviert ist. Dort steht nun die Nummer des nächsten von dieser Datei belegten Daten-Clusters.

Nun springt man in den FAT-Platz, der nun wieder für diesen Cluster reserviert ist usw., bis man auf einen Platzeintrag mit dem Wert &HFFF trifft. Dieser Wert stellt die Endmarkierung einer Datei dar.

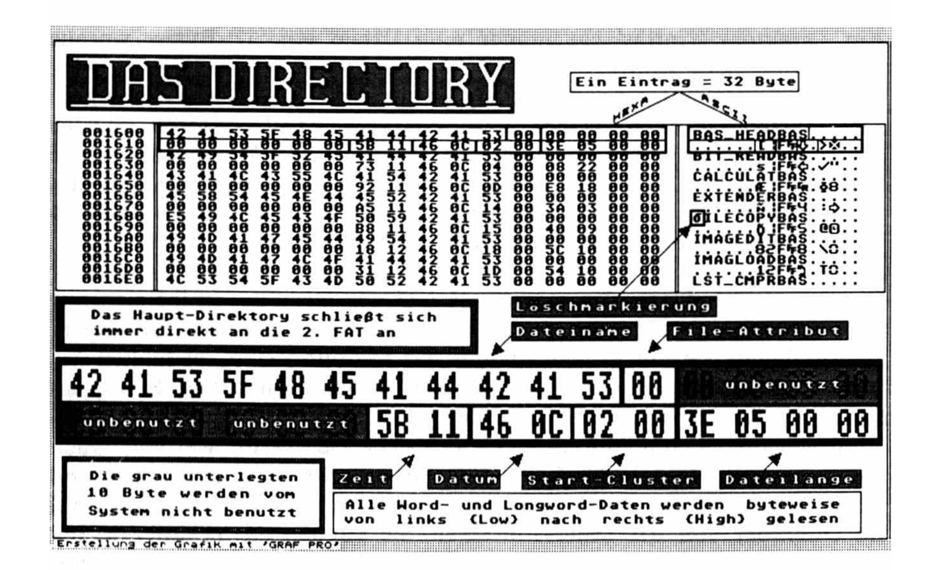
D.h., daß der Cluster, in dessen FAT-Platz der Wert &HFFF steht, der letzte dieser bestimmten Datei ist.

Um jetzt feststellen zu können, welches Byte innerhalb dieses letzten Daten-Clusters das letzte Byte der Datei ist, wird die Anzahl der gelesenen Cluster mit der im Direktory eingetragenen Dateilänge verglichen. Ist die Datei z.B. 2466 Byte lang, muß also das 418. Byte (2466-2*1024) das letzte Byte der Datei sein.

Damit nicht genug! Daß ein Cluster zwei Sektoren umfaßt, wissen Sie bereits. Wenn eine einseitige Diskette 720 Sektoren hat, denkt man sich, daß demnach also 360 Cluster vorhanden sein müßten! Weit gefehlt! Im Normalfall hat eine 720-Sektoren-Diskette nur 351 Daten-Cluster.

Das liegt daran, daß die ersten 18 Sektoren (normalerweise!) für Boot-Sektor, FATs und Haupt-Direktory ausgenommen sind. Aus diesem Grund sind die ersten beiden FAT-Plätze mit &HF7FFFF belegt. So einfach ist das!

Um das Chaos noch perfekt zu machen, müssen auch die Einträge der FAT-Plätze verdreht gelesen werden (MS-DOS). Es ist schon kompliziert genug, aus jeweils drei Byte für zwei FAT-Plätze immer die Halb-Bytes zu selektieren. Daß man nun auch noch die drei einzelnen Nibbles (je 4 Bit) auch noch vertauschen muß, setzt dem Ganzen die Krone auf. Das kann passieren, wenn das Betriebs-System eines bestehenden Computers einfach (und das so schnell wie möglich) aufeinen neuen Computer "aufgesetzt" wird (MS-DOS: Microsoft-Disk-Operating.System -> TOS: Tramiel-Operating-System).



Directory

Ein Directory (engl.: Adreßbuch) beinhaltet üblicherweise die Namen aller Dateien, die jemals (seit der letzten Formatierung!) auf dieser Diskette angelegt wurden. Es sei denn, der Dateiname wurde per Monitor komplett gelöscht. Es ist grundsätzlich 7 Sektoren groß und der erste Direktory-Sektor liegt üblicherweise direkt hinter dem letzten Sektor der zweiten (bzw. letzten) FAT.

Wurden Dateien gelöscht (z.B. Desktop-Papierkorb oder KILL-Funktion), wird der erste Buchstabe des Dateinamens durch das SIGMA-Zeichen (&HE5E5) ersetzt und der entsprechende FAT-Eintrag mit einer Null-Byte-Folge gelöscht. Die Länge dieser Folge ist ist von der Größe der gelöschten Datei abhängig.

War die Datei z.B. 5000 Bytes groß, so hat sie (INT(5000/512)+1)) 10 Sektoren an Speicherplatz belegt (eine Datei beginnt immer mit dem ersten Byte des nächstfreien Sektors). Wie oben beschrieben, belegt ein Sektor 1.5 Bytes in der FAT, woraus sich eine Null-Byte-Folge von (10*1.5) 15 Bytes ergibt.

Im Anschluß an die jeweiligen Datei- und Ordnernamen enthält ein Direktory-Eintrag zusätzliche Informationen über Datum und Uhrzeit der Datei-Erstellung, über Datei-Atrribute sowie über Dateilänge und Start-Cluster.

Ein Direktory-Eintrag hat immer eine Länge von 32 Bytes, woraus sich 16 mögliche Einträge pro Direktory-Sektor ergeben. Wie Sie wissen, kann man auf einer Direktory-Ebene max. 112 Dateien ablegen. Jetzt wissen Sie auch, woher diese Zahl kommt: 7 Sektoren mal 16 Einträge sind max. 112 Einträge pro Direkory. Sie können allerdings Ordner anlegen, die dann wieder jeder 112 weitere Einträge aufnehmen können. Es bestehen dann außer dem Hauptverzeichnis noch so viele Unter-Verzeichnisse, wie Ordner vorhanden sind.

Die Lage eines Unter-Verzeichnisses auf der Diskette ergibt sich aus dem im Direktory hinter dem Ordner-Namen eingetragenen Start-Sektor. Wenn Sie ein großer Freund der Ordnung sind, bedenken Sie bitte, daß für jeden angelegten Ordner 7 Sektoren (3584 Bytes) an Speicherplatz "verschwendet" werden und einmal gemachte Ordner-Einträge im Haupt-Direktory nicht wieder entfernt werden, auch wenn man den Ordner löscht. Hatten Sie z.B. 10 Ordner angelegt und alle wieder gelöscht, bleibt trotzdem nur noch eine freie Eintragszahl im Direktory von 102 Datei-Einträgen. Sie können das revidieren, indem Sie die - nun überflüssigen - Ordner-Einträge im Direktory mit Null-Bytes überschreiben. Dabei müssen allerdings alle Einträge, die hinter der Löschung liegen, um eine Position nach vorn verschoben werden, damit keine Lücken im Direktory entstehen. Das System durchsucht bei der Datei-Suche nur bis zum ersten Null-Eintrag.

Dateinamen hinter einer Null-Lücke werden dann also nicht mehr gefunden und können auch nicht mehr auf dem Direktory (oder durch FILES, DIR$, FILESELECT etc.) sichtbar gemacht werden. Dies ist übrigens eine interessante Möglichkeit, versteckte Dateien zu produzieren.

Var=XBIOS(18,L:Adresse,L:0,L:Serie,Typ,Tos_flag)-PROTOBT-

Schreibt die angegebenen Daten in einen ab Adresse liegenden Boot-Sektor, erzeugt die 2-Byte-Checksumme und trägt sie in die letzten zwei Bytes des Puffers ein.

Adresse	**Startadresse eines 512-Byte-Puffers, in dem der vorbereitete Boot-Sektor liegt.**
Serie	**24-Bit-Wert, der die Seriennummer der Diskette darstellt. Wird ein Wert größer &HFFFFFF übergeben, so wird ein zufälliger Wert erzeugt.**
Typ	**Beschreibt den Disk-Typ (Media):** 0 = 40 Tracks - einseitig (252) 1 = 40 Tracks - zweiseitig (253) 2 = 80 Tracks - einseitig (248) 3 = 80 Tracks - einseitig (249)

	Abhängig von dem hier übergebenen Wert wird von der Funktion der Media-Descriptor erzeugt und eingetragen.
Tos_flag	Gibt an, ob es sich um eine ausführbaren Boot-Sektor inkl. Boot-Programm handelt (z.B. Disketten-TOS). 0 = Diskette ohne Boot-Programm 1 = Diskette mit Boot-Programm Wird in einem der letzten drei Parameter der Wert -1 übergeben, so bleibt der entsprechende Eintrag im Sektor unverändert.

Var=XBIOS (64,Blit%) *-blitmod-*

-Nur mit Blitter-TOS-

Fragt den Blitter-Status ab, bzw. schaltet den Blitter ein oder aus.

Blit%:

-1	In Var wird der TRUE (-1) oder FALSE (0) geliefert (an/aus)
0	Blitter ausschalten
1	Blitter anschalten

21.5.2 Disketten formatieren

Wer möchte nicht gern seine Disketten selbst formatieren können. Es folgt nun eine ausführliche Beschreibung jedes einzelnen Schrittes, der zu einer Formatierung notwendig ist. Auch wenn Sie später doch lieber die Desktop-Formatier-Routine verwenden, können Sie so wichtige Einblicke in die Organisation einer Diskette gewinnen.

Legen Sie bitte eine unformatierte Leerdiskette in Laufwerk A. Ich gehe hier nun davon aus, daß diese Diskette einseitig mit 80 Tracks und 9 Sektoren pro Track standardmäßig formatiert werden soll. Möchten Sie Ihre Disketten mit Spezialformaten versehen, bleibt das später Ihrer Phantasie und Experimentierfreude überlassen.

```
' Zuerst wird die Diskette rein physikalisch in Tracks und Sektoren
' eingeteilt. Damit das auch professionell aussieht, verwandle
' ich den Bildschirm in ein "Desktop".
'
Deffill ,2,4                  !-----.
Pbox 0,17,639,399                   !
Print At(36,1);"FORMATIEREN"        !
Deffill ,0,0                        !
Pbox 150,100,490,299                !
Pbox 153,103,487,296                !- Screen-Aufbau
Pbox 154,104,486,295                !
Print At(24,10);"Formatieren"       !
Print At(24,12);"In Arbeit .."      !
Box 180,230,460,248                 !
Deffill ,2,2                  !-----'
'
X$=Space(8000)              ! Arbeitspuffer vorbereiten
For I%=0 To 79              ! 80 Tracks...
  Void Xbios(10,L:Varptr(X$),L:0,0,9,I%,0,1,L:&H87654321,&HE5)
  ' ...der Seite 0 mit je 9 Sektoren auf Disk A: formatieren
  Pbox 180,230,180+280/80*(I%+1),248
Next I%
Deffill ,2,4
Pbox 0,17,639,399           ! Screen löschen
'
' Um eine frisch formatierte Diskette verwenden zu können, muß vorher
' der Boot-Sektor und die FAT auf der Diskette eingerichtet werden.
'
' Das Wichtigste ist dabei, zu wissen, wie ein Boot-Sektor aufgebaut
' ist. Bei allen folgenden Word-Werten ist zu beachten, daß sie im
' MS-DOS-Format angelegt sind, d.h. daß die High- und Low-Bytes
' miteinander vertauscht sind (z.B. &HB16E wird als &H6EB1 gelesen).
'
Boot$=Mki$(0)               ! Branch
'
' Die ersten 2 Bytes sind ein Zeiger auf ein Boot-Programm, das
' dann ggfs. ab dem 31. Byte des Boot-Sektors zu installieren
' ist (z.B. bei TOS-Boot-Disketten oder bei Disk-Viren (!?)).
' Da es sich hier um eine Standard-Disk handelt, lasse ich
' diesen Zeiger unberücksichtigt - also Null.
```

```
'
Boot$=Boot$+String$(6,"N") ! 'Loader'
'
'  Die Bytes 3-8 sind ein sogenannter "Loader". Der Inhalt dieser
'  6 Bytes ist nur dann von Interesse, wenn die ersten zwei Byte
'  auf ein Boot-Programm zeigen.
'  Hier sind diese 6 Bytes mit N belegt, wie es die Desktop-
'  Formatier-Routine ebenfalls tut.
'
Boot$=Boot$+"000"          ! Serial
'
'  Die nächsten 3 Byte (9, 10 u. 11 = 24 Bit) gelten als Puffer für
'  den an XBIOS(18) übergebenen 24-Bit-Wert (ggfs. Zufallszahl),
'  der als Seriennummer interpretiert wird.
'
Boot$=Boot$+Mki$(&H2)      ! BPS -> MS-DOS (entspricht &H200)
'
'  Byte 12 und 13 enthalten als Word die Anzahl der Bytes pro Sektor.
'  Dies ist standardmäßig der Wert 512 (&H200 -> in MS-DOS: &H2 s.o.).
'  Da ich mit diesem Wert noch nicht experimentiert habe, kann ich
'  nicht sagen, ob es möglich ist, auch mehr oder weniger als
'  512 Bytes pro Sektor zuzuordnen.
'
Boot$=Boot$+Chr$(2)        ! 'SPC'
'
'  Byte 14 enthält als Byte die Anzahl der Sektoren pro Cluster.
'  Dies ist üblicherweise der Wert 2. Auch mit diesem Wert habe
'  ich noch keine Experimente angestellt.
'
Boot$=Boot$+Mki$(&H100)    ! RES -> MS-DOS (entspricht &H1)
'
'  Byte 15 und 16 enthält als Word die Anzahl reservierter Sektoren.
'  Üblich: 1 (&H1 -> in MS-DOS: &H100 s.o.). Damit ist der Boot-Sektor
'  selbst gemeint.
'
Boot$=Boot$+Chr$(2)        ! FAT
'
'  Byte 17 ist wieder ein Byte und enthält die Anzahl der FATs
'  (File-Allocation-Tables). Üblich: 2. Jede Eintragsänderung
'  in der 1. Haupt-FAT wird auch gleichzeitig in der Reserve-FAT
'  durchgeführt. Ist evtl. eine FAT beschädigt, kann die Diskette
'  trotzdem noch gelesen werden, wenn man die unbeschädigte
'  FAT auf die beschädigte kopiert.
'
Boot$=Boot$+Mki$(&H7000) ! 'DIR' -> MS-DOS (entspricht &H70)
'
'  Byte 18 und 19 sind wieder ein MS-DOS-Word, das die max. Anzahl
'  der Einträge pro Direktory (inkl. Ordner) angibt.
'  Üblich: 112 (&H0070 -> in MS-DOS: &H7000). Wenn Sie einen anderen
'  Wert verwenden möchten, sollten Sie darauf achten, daß er durch
'  16 teilbar ist.
'
Boot$=Boot$+Mki$(&HD002) ! 'SEC' -> MS-DOS (entspricht &H2D0)
'
'  Bytes 20 und 21 enthalten als MS-DOS-Word die Gesamtanzahl der
'  Sektoren auf der Diskette. Hier 720 (&H2D0 -> in MS-DOS: &HD002)
```

```
'                                            Einseitig / Zweiseitig
'  80 Tracks je  9 Sektoren pro Seite =      720    /     1440
'  81 Tracks je  9 Sektoren pro Seite =      729    /     1458
'  82 Tracks je  9 Sektoren pro Seite =      738    /     1476
'  80 Tracks je 10 Sektoren pro Seite =      800    /     1600
'  81 Tracks je 10 Sektoren pro Seite =      810    /     1620
'  82 Tracks je 10 Sektoren pro Seite =      820    /     1640
'
Boot$=Boot$+Chr$(248)      ! 'MEDIA'
'
'  Das Byte 22 ist der sogenannte Media-Descriptor. Er ist bei
'  einseitigen 80-Track-Disketten auf 248 und bei zweiseitigen
'  80-Track-Disketten auf 249 festgelegt und wird ggfs. durch
'  XBIOS(18) nachgetragen.
'
Boot$=Boot$+Mki$(&H500)  ! SPF -> MS-DOS (entspricht &H5)    '
'  Byte 23 und 24 gibt als MS-DOS-Word die Anzahl der Sektoren
'  pro FAT an. Üblich: 5 (&H5 -> in MS-DOS: &H500).
'  Sie können auch weniger als 5 FAT-Sektoren festlegen, solange
'  für jeweils einen Diskettensektor eineinhalb Bytes (12 Bits)
'  Platz darin enthalten sind (siehe 21.5.1 "Aufbau einer Diskette")
'
Boot$=Boot$+Mki$(&H900)  ! SPT -> MS-DOS (entspricht &H9)    '
'  Die Bytes 25 und 26 sind ein MS-DOS-Word, das die Anzahl an
'  Sektoren pro Track enthält. Üblich: 9 (&H9 -> in MS-DOS: &H900).
'  Es sind auch 10 Sektoren pro Track möglich. Mehr können vom
'  GEMDOS nicht verarbeitet werden.
'
Boot$=Boot$+Mki$(&H100)  ! SIDE -> MS-DOS (entspricht &H1)
'
'  Bytes 27 und 28 sind wieder ein MS-DOS-Word mit der Anzahl der
'  Seiten der Diskette.
'          Einseitig : 1 (&H1 ST -> in MS-DOS: &H100)
'          Zweiseitig: 2 (&H2 ST -> in MS-DOS: &H200)
'
Boot$=Boot$+Mki$(0)        ! HID
'
'  Das letzte bedeutungsvolle Word gibt in den Bytes 29 und 30
'  die Anzahl versteckter Sektoren an. Standard ist 0.
'  Versteckte Sektoren werden bei der Platzvergabe an die
'  Dateien durch das GEMDOS "übersehen".
'
Boot$=Boot$+String$(30,"N")+String$(12,0)
Boot$=Boot$+String$(3,245)+Chr$(&HFE)+Chr$(&H4F)
Boot$=Boot$+Chr$(0)+Chr$(1)+Chr$(2)+Chr$(&HF7)
Boot$=Boot$+String$(22,"N")+String$(12,0)
Boot$=Boot$+String$(3,245)+Chr$(&HFB)+String$(391,&HE5)
'
'  Die Daten dieses Blocks bedürfen keiner Erklärung. Es handelt
'  sich um die Bytes 31 bis 510, die bei Standard-Disketten keine
'  erkennbare Bedeutung haben.
'  Bei ausführbaren Disketten (siehe Word 1) ist hier das Boot-Programm
'  zu finden.
'  Dieser Block ist hier so aufgebaut, wie es die Formatier-Routine
'  des Desktops auch tut. Dabei besteht auch zwischen ein- und
'  zweiseitigen Disketten kein Unterschied. Dieser Block ist immer
'  gleich aufgebaut.
'
```

```
Boot$=Boot$+Mki$(&H0)    ! 'CHECKSUM'
'
'  Zum Schluß wird noch ein Word als Puffer für die
'  Check-Summe angehängt. Dieser Wert wird von XBIOS(18) dort
'  eingetragen.
'
' Die Check-Summe ergibt sich daraus, daß alle Words des Boot-Sektors
' addiert werden (inkl. Check-Summe). Soll der Boot-Sektor ausführbar
' sein, ist die Check-Summe als Ausgleichs-Word für die Summe aller
' übrigen 205 Words anzusehen. Das letzte Word (LO-Word) der gesamten
' Summe (inkl. Check-Summe) muß dann den Hexwert $1234 ergeben.
'
' Ein Boot-Programm muß mit RTS enden und
' darf keine GEM-Aufrufe enthalten.
'
'  Wenn Sie es sich einfach machen wollen, können Sie natürlich auch
'  einen fertigen Boot-Sektor von einer "sauberen" Diskette
'  durch XBIOS(8) oder BIOS(4) in einen Puffer laden und die gewünschten
'  Daten einzeln eintragen.
'  Für den XBIOS(8)- bzw. XBIOS(9)-Aufruf liegt der Boot-Sektor in
'  Sektor 1 von Track 0 auf Seite 0. Für den BIOS(4)-Aufruf liegt er
'  im logischen Sektor 0.
'
'  Den fertigen Boot-String schicke ich nun durch die
'  Boot-Sektor-"Waschanlage" XBIOS(18):
'
Void Xbios(18,L:Varptr(Boot$),L:&H1FFFFFF,2,0)
'
'  Dabei wird durch den Wert &H1FFFFFF gesagt, daß ein Zufallswert
'  als Seriennummer dienen soll. Durch 2 wird der Disktyp übergeben
'  (2=80 Tracks einseitig) und durch die Null gilt der Boot-Sektor
'  als nicht ausführbar.
'
'  Der Boot-Sektor ist fertig.
'
'  Der nächste Schritt ist notwendig, um die Diskettenverwaltung
'  des GEMDOS auf den neuesten Stand zu bringen. Sonst kann es
'  passieren, daß die Speicherplatz-Ermittlung (siehe GEMDOS(54))
'  nicht mehr funktioniert.
'
Void Bios(7,0)             ! BIOS-Parameter-Block holen
'
'  Da die Diskette komplett formatiert wurde, muß der neue Boot-Sektor
'  auf der Diskette abgelegt und anschließend die FATs erzeugt werden.
'
'  Der Boot-Sektor wird einfach entweder mit:
'
Void Xbios(9,L:Varptr(Boot$),L:0,0,1,0,0,1)
'
'  oder mit
'  VOID BIOS(4,3,L:VARPTR(Boot$),1,0,0)
'  auf die Diskette in Laufwerk A geschrieben.
'
'  Die Erfahrung zeigt, daß es ratsamer ist, den Boot-Sektor über
'  das XBIOS auf die Diskette zu schreiben. In sehr seltenen
'  Fällen kann es bei der BIOS(4)-Variante zu Störungen kommen.
'  Warum das so ist, kann ich Ihnen leider nicht verraten.     '
'  Aus oben genanntem Grund wird noch einmal der BIOS-Parameter-Block
'  angefordert.
```

```
'
Void Bios(7,0)            ! BIOS-Parameter-Block holen
'
'  Nun zur FAT.
'  Der erste FAT-Sektor besteht bei einer neu formatierten Diskette aus
'  lediglich den ersten drei Byte, die immer &HF7FFFF enthalten und den
'  restlichen 509 Null-Bytes. Die übrigen Sektoren derselben FAT
'  bestehen ausschließlich aus Null-Bytes. Dies ist wichtig, wenn
'  beim Formatieren der Diskette durch XBIOS(10) kein Null-Byte zum
'  Füllen verwendet wurde.
'  Die erste FAT liegt immer direkt hinter dem
'  Boot-Sektor und direkt daran schließt sich dann die zweite FAT an
'  (siehe 21.5.1 "Aufbau einer Diskette").
'
'  Im obigen Fall müssen also 2 mal 5 Sektoren für die beiden FATs
'  hergerichtet werden.
'
A$=Mki$(&HF7FF)+Chr$(&HFF)+String$(2557,0) ! 1.FAT
A$=A$+A$                                   ! 1. + 2.FAT
Void Bios(4,3,L:Varptr(A$),10,1,0)         ! FATs schreiben
'
'  Wie unter "Aufbau einer Diskette" bereits erwähnt
'  finden Sie immer direkt hinter der 2. (bzw. letzten) FAT
'  das Haupt-Direktory der Diskette, das üblicherweise 7 Sektoren
'  umfaßt. Aus dieser Sektorenfolge ergibt sich, daß bei
'  Standard-Disketten der erste Daten-Sektor auf Sektor 18
'  liegt (1 Boot-Sektor + 2*5 FAT-Sektoren + 7 Direktory-Sektoren).
'
'        Wer dies weiß, hat dadurch eine sehr einfache Methode
'        zur Verfügung, eine Diskette komplett zu "löschen":
'
'        A$=Mki$(&HF7FF)+Chr$(&HFF)+String$(2557,0)
'        A$=A$+A$+String$(512*7,0)
'        Void Bios(4,3,L:Varptr(A$),17,1,0)
'
'        Durch diesen simplen Dreizeiler werden die FATs und
'        das Direktory mit einem "unschuldigen" Standard-Block
'        von 17 Sektoren einfach überschrieben. Voraussetzung
'        dazu ist allerdings, daß die Diskette im Standard-
'        (Desktop-)Format schon korrekt formatiert wurde.
'        Das Diskettenformat (ein- oder doppelseitig) ist dabei
'        egal. Die vorher auf der Diskette vorhandenen Datei-Daten
'        werden nun durch neue Speicher-Vorgänge einfach
'        überschrieben.
'
'        Sollte die Diskette ein außergewöhnliche Anordnung
'        der FATs und des Direktories haben, müßten die A$=-Zeilen
'        entsprechend geändert werden.
'
'  Wenn bis jetzt alles glatt gegangen ist, dann ist die
'  in Laufwerk A befindliche Diskette standardmäßig
'  komplett und korrekt formatiert.
'  Es folgt nun eine Routine, die Ihnen den kompletten
'  Boot-Sektor und die Disketten-Daten auf dem Bildschirm
'  anzeigt. Dazu holen wir uns den neuen Boot-Sektor von der
'  Diskette und übergeben ihn an diese Prozedur.
'
Void Bios(4,2,L:Varptr(Boot$),1,0,0)
@Showboot(Boot$)
```

```
'
' Um zu überprüfen, ob auch der Datei-Zugriff funktioniert,
' speichern Sie bitte dieses Formatierungsprogramm durch
' Save,A ab und laden es durch Merge wieder in den Speicher.
' Das müßte reibungslos geklappt haben.
'
```

```
Bootsektor 1   *alt*
1234567890123456
```

```
Alert 1,"Freier Speicher :| |     "+Str$(Dfree(0))+"| ",1,"Return",A%
'
' Als krönenden Abschluß lassen wir uns noch den freien
' Speicherplatz der neuen Diskette ausgeben.
'
Procedure Showboot(Bootsek$)
  Local A$,A%,J%,I%,Byte%
  A$=Mki$(&HA000)+Mki$(&H2009)+Mki$(&H4E75) !--.
  Void Fre(0)                                  !
  A%=Varptr(A$)                                !
  Lpoke Intin,Lpeek(C:A%()+4)                  !- 8x8-Font
  Dpoke Contrl+2,0                             !  einschalten
  Dpoke Contrl+6,2                             !
  Dpoke Contrl+10,102                          !
  Vdisys 5                                    !--'
  Graphmode 1                                 !--.
  Deffill ,0,0                                 !
  Pbox 3,301,225,395                           !
  Box 3,301,226,396                            !
```

```
    Pbox 3,4,162,298                          !- Screen-
    Box 3,4,163,299                           !  Aufbau
    Print At(2,2);"Boot-Sektor ";Str$(Fl%)    !
    Print At(2,3);"------------------"        !
    Print At(2,4);"1234567890123456"          !
    Print At(2,5);"------------------"      !--'
    For J%=0 To 31                          ! 32 Zeilen
      Print At(2,Crslin);
      For I%=0 To 15                        ! Zu je 16 Bytes
        Byte%=Peek(Varptr(Bootsek$)+I%+J%*16) ! Zeichen lesen
        If Byte%=0                          ! Byte =0?
          Out 5,Asc(".")                    ! Dann Punkt schreiben
        Else                                ! Byte >0
          Out 5,Byte%                       ! Zeichen ausgeben
        Endif
      Next I%
      Print "¦";J%+1
    Next J%
    Print At(2,39);"Bytes pro Sektor    : ";         !-.
    Print Cvi(Mid$(Bootsek$,13,1)+Mid$(Bootsek$,12,1))  !
    Print At(2,40);"Sektoren pro Cluster: ";            !
    Print Asc(Mid$(Bootsek$,14,1))                      !
    Print At(2,41);"Reservierte Sektoren: ";            !
    Print Cvi(Mid$(Bootsek$,16,1)+Mid$(Bootsek$,15,1))  !
    Print At(2,42);"Anzahl der FATs     : ";            !
    Print Asc(Mid$(Bootsek$,17,1))                      !
    Print At(2,43);"max. Einträge in Dir: ";            !
    Print Cvi(Mid$(Bootsek$,19,1)+Mid$(Bootsek$,18,1))  !
    Print At(2,44);"Anzahl der Sektoren : ";            !
    Print Cvi(Mid$(Bootsek$,21,1)+Mid$(Bootsek$,20,1))  !- Disk-
    Print At(2,45);"Media Descriptor    : ";            !  Daten
    Print Asc(Mid$(Bootsek$,22,1))                      !  lesen
    Print At(2,46);"Sektoren pro FAT    : ";            !  und
    Print Cvi(Mid$(Bootsek$,24,1)+Mid$(Bootsek$,23,1))  !  ausgeben
    Print At(2,47);"Sektoren pro Track  : ";            !
    Print Cvi(Mid$(Bootsek$,26,1)+Mid$(Bootsek$,25,1))  !
    Print At(2,48);"Seiten pro Disk     : ";            !
    Print Cvi(Mid$(Bootsek$,28,1)+Mid$(Bootsek$,27,1))  !
    Print At(2,49);"versteckte Sektoren : ";            !
    Print Cvi(Mid$(Bootsek$,30,1)+Mid$(Bootsek$,29,1))!-'
    Void Fre(0)                             ! Garbage Collection
    A%=Varptr(A$)                           !----.
    Lpoke Intin,Lpeek(C:A%()+8)                 !- 8x16-Font ein
    Vdisys 5                                !----'
  Return
```

Mit dieser "Showboot"-Routine haben Sie so ganz nebenbei auch die Möglichkeit, Ihre Disketten auf Disk-Viren zu untersuchen. Ist nämlich auf einer normal erscheinenden Diskette im Boot-Sektor ein Boot-Programm untergebracht und die ersten zwei Bytes des Boot-Sektors sind nicht Null, so ist höchste Vorsicht angebracht. Sollte es tatsächlich ein Virus sein, so kann es sein, daß schon ein Großteil Ihrer Disketten verseucht ist. Manche Viren sind so hartnäckig, daß sie sich bei jedem Diskettenzugriff, der intern BIOS(7) auslöst, selbstständig wieder in den Boot-Sektor kopieren.

Um einen solchen Disk-Virus loszuwerden, benötigen Sie eine garantiert "gesunde" Diskette, die bei Systemstart in Laufwerk A liegt. Nun können Sie sich mit dem eben erworbenen Wissen an die Säuberung des Boot-Sektors bzw. der Boot-Sektoren machen. Sobald jedoch ein Virus unbemerkt auf einer Diskette überlebt und Sie haben diese Diskette bei Systemstart in A liegen, geht das ganze wieder von vorn los. Das eben gesagte gilt wohlgemerkt nur für die Disk-Viren, die sich über das Boot-Programm einschleusen.

Viren, die in irgendwelche Programme eingebaut wurden, sind dagegen um ein Vielfaches hartnäckiger und wesentlich schwerer zu finden. Meist ist dieser Virentyp auch um Einiges bösartiger (zerstörte Dateien, beschädigte Schreib/Leseköpfe ect. etc.). Bei Computer-Viren aller Art (es sei denn, es sind Spaß-Viren), gilt immer höchste Alarmstufe, da ja evtl. nicht nur Sie davon betroffen sein können, sondern mit jeder infizierten Diskette und jedem infizierten Programm der Virus weitergegeben wird. Welche verheerenden Wirkungen (auch finanzieller Art) das haben kann, mußten schon einige Software-Firmen und Anwender bitter erfahren. An Virus-Produzenten zu appellieren, ihre pubertären Ambitionen im Zaum zu halten, hat wohl wenig Sinn.

Deshalb hilft da nur die Wachsamkeit jedes Einzelnen.

Var=XBIOS(19,L:Adresse,L:0,Laufwerk,Sektor,Track,Side,Anzahl)
-FLOPVER-

Vergleicht (verifiziert) den/die angegebenen Disketten-Sektor(en) mit dem ab Adresse liegenden Speicherbereich. Die Parameter sind dieselben wie bei XBIOS(8) (Flopread).

Stimmen die Diskettendaten mit den Speicherdaten nicht überein, so steht ab Adresse eine mit einem Null-Byte abgeschlossene Word-Liste der fehlerhaften Sektor-Nummern.

Var=XBIOS(20) *-SCRDMP-*

Es wird eine Hardcopy ausgedruckt. Entspricht HARDCOPY in V2.xx, <Alternate><Help> bzw. der Desktop-Funktion "Bildschirm drucken".

HARDCOPY in V3.0 funktioniert geringfügig anders (siehe dort).

Var=XBIOS(21,Modus,Frequenz) *-CURSCONF-*

Mit dieser Funktion kann der TOS-Cursor beeinflußt werden.

Modus	0 = Cursor aus	1 = Cursor an
	2 = Cursor blinken	3 = Cursor nicht blinken
	4 = Neue Frequenz setzen	5 = Aktuelle Frequenz holen
Rate	(Nur wenn Modus = 4) Gibt in 1/70 (Hires) bzw. 1/50 (COLOR) Sekunden die Blinkfrequenz des Cursors an. Ein Wert von z.B. 35 läßt den Cursor in Hires zweimal in der Sekunde blinken.	

Bei Modus = 5 wird die aktuelle Frequenz in Var zurückgegeben.

Var=XBIOS(22,L:Adresse) *-SETTIME-*

Entspricht den GEMDOS-Funktionen 43 und 45. Die dort getrennten Angaben sind hier zu einem Longword kombiniert, das ab Adresse liegen muß. Das Datum liegt dabei im HI-Word des Longwords und die Zeitangabe im LO-Word (Format s. GEMDOS(43)/GEMDOS(45)).

Var=XBIOS(23) *-GETTIME-*

Liefert in Var einen 4-Byte-Wert. Diese Funktion entspricht den GEMDOS-Funktionen 42 und 44 (siehe dort sowie unter XBIOS(22)).

Var=XBIOS(24) *-BIOSKEYS-*

Wurde mit XBIOS(16) (Keytable) die Tastaturbelegung geändert, kann hiermit der Urzustand wiederhergestellt werden, falls Sie nicht die Belegung so geändert haben, daß die normale Tastaturbelegung außer Kraft gesetzt ist und man keine "vernünftigen" Zeichen mehr eingeben kann (siehe Beispiel unter XBIOS(16)).

Var=XBIOS(25,Bytes,L:Adresse) *-IKBDWS-*

Sendet einen String von Bytes-1 Bytes an den Tastaturprozessor (IKBD = Intelligent Keyboard). Der String wird ab Adresse gelesen. Ausführliche Erläuterungen hierzu finden Sie unter OUT.

Var=XBIOS(26,Mfp_index) *-JDISINT-*

Der Interrupt im MFP-Vektor mit der Nummer Mfp_index kann hiermit gesperrt werden.

Der MFP-Chip fungiert im ST als Interrupt-Controller. In einem Vektor sind hier 16 Interrupt-Routinen aufgeführt, die nacheinander bei jedem Interrupt ausgeführt werden.

Mfp_index:

0	Centronics Strobe.
1	RS232 DCD.
2	RS232 CTS (RS232-Empfänger bereit, Daten senden)
3	Nicht belegt.
4	RS232-Baud-Raten-Generator (Timer D) (frei zur eigenen Belegung, siehe XBIOS(31))
5	Timer C (Systemtakt setzen, Sound-Verarbeitung).
6	ACIA-Kontrolle (Tastatur, MIDI, Joystick-Ports).
7	Disk-Controller, DMA.
8	HBL-Zähler (Timer B).
9	RS232-Output-Error (Transmitter-Status in RS232-Parameter-Block).
10	RS232-Output-Puffer leer (Sendebereitschaft, Sende-Register setzen).
11	RS232-Input-Error (über High water oder falsche Parity -> Empfangs-Status löschen)
12	RS232-Input-Puffer leer (unter Low water) (Data-Empfang zulassen)
13	System Clock (Timer A - reserviert).
14	RS232-Ring-Indicator (Ring-Zeiger).
15	Monochrom Monitor Detect.

Var=XBIOS(27,Mfp_index) *-JENABIN-*

Der Interrupt im MFP-Vektor mit der Nummer Mfp_index wird wieder zugelassen (siehe XBIOS(26)).

Var=XBIOS(28,Byte,Register) *-GIACCES-*

Ermittelt bzw. verändert die Inhalte der Sound-Register. (Bedeutung der Register siehe XBIOS(32)).

Byte	Byte-Wert, der in das mit Register angegebene Sound-Register geschrieben werden soll. Byte wird nur berücksichtigt, wenn in Register das Bit 7 gesetzt ist.
Register	Die unteren 4 Bits dieses Byte-Wertes bestimmen das gewünschte Sound-Register (0 - 15). Ist das höchste Bit (Bit 7) dieses Bytes gesetzt, wird der Wert Byte in das angegebene Sound-Register 0+(2^7) bis 15+(2^7) geschrieben. Ist das Bit 7 nicht gesetzt, so wird der aktuelle Inhalt des Registers in Var geliefert. Vorsicht: Register 14 und 15 haben nichts mit der Sound-Verarbeitung zu tun, sondern sind für die Floppy da (Funktion unbekannt, siehe XBIOS(29)/XBIOS(30)).

Beispiel:

```
For I%=0 To 15
  Print "Inhalt des Sound-Registers ";I%;" = ";
  Print Xbios(28,0,I% And &HFF)
Next I%
```

Var=XBIOS(29,Vektor) *-OFFGIBIT-*

Sound-Chip-(Floppy/DMA)-Port-A-Bits setzen. Vektor ist ein 7-Bit-Vektor, dessen Bits angeben, welche Funktion angeschaltet werden soll.

Vektor:

Bit 0	Disk-Seite 0.
Bit 1	Disk-Station A an.
Bit 2	Disk-Station B an.
Bit 3	RS232 RTS (RequestToSend) an.
Bit 4	RS232 DTR (DataTerminalReady) an.
Bit 5	Centronics Strobe Busy.
Bit 6	GPO GeneralPurposeOutput-Pin an.

Beispiel:

```
For I%=1 To 10
   Void Xbios(29,&X110)
   Print At(10,10);"Diskette einlegen"
   Pause 5
   Void Xbios(30,&X10)
   Print At(10,10);"                 "
   Pause 5
Next I%
```

Var=XBIOS(30,Vektor) *-ONGIBIT-*

Soundchip-(Floppy/DMA)-Port-A-Bits löschen. Vektor ist ein 7-Bit-Vektor, dessen Bits angeben, welche Funktion ausgeschaltet werden soll.

Vektor:

Bit 0	Disk-Seite 1.
Bit 1	Disk-Station A aus.
Bit 2	Disk-Station B aus.
Bit 3	RS232 RTS (RequestToSend) aus.
Bit 4	RS232 DTR (DataTerminalReady) aus.
Bit 5	Centronics Strobe Not Busy.
Bit 6	GPO GeneralPurposeOutput-Pin aus.

Beispiel siehe unter XBIOS(29).

Var=XBIOS(31,Timer,Control,Data,L:Adresse) *-XBTIMER-*

Installiert eine MFP-Timer-Interrupt-Routine und startet den MFP-Timer bzw. verändert das Control- und Data-Register des betreffenden Timers (siehe XBIOS(26).

Timer	Gibt den gewünschten MFP-Timer an: Timer A = 0/B = 1/C = 2/D = 4
Control	Wert, der in das Kontroll-Register des betreffenden Timers geschrieben wird.
Data	Wert, der in das Data-Register des betreffenden Timers geschrieben wird. Die Nutzung dieser beiden Parameter setzt eine genaue Kenntnis der Hardware-Bedingungen des MFP 68901-Chips voraus.
Adresse	Wird -1 übergeben, bleibt die alte Timer-Routine erhalten. Ist Adresse größer Null, so wird dies als Startadresse der neuen Routine gewertet.

Var=XBIOS(32,L:Adresse) *-DOSOUND-*

Hiermit kann ein vorgegebener Sound-String auf Interrupt-Ebene abgearbeitet werden. Der SOUND- und WAVE-Befehl benötigt eine gewisse Zeitspanne, um den eingestellten Sound zu verarbeiten. XBIOS(32) kümmert sich dagegen nicht um das laufende Programm, sondern arbeitet im Hintergrund auf Interrupt-Ebene selbstständig weiter. Es ist also möglich, beliebige Programmabläufe mit Geräuschen und Klängen zu untermalen. Dazu sind in den ersten 14 Registern des Sound-Chips nach Bedarf bestimmte Werte zu setzen, die sich die Funktion aus dem String abholt. Außerdem können über drei spezielle Kommandos, die in den String integriert werden, drei weitere Funktionen bestimmt werden, die dann von der Dosound-Interrupt-Routine ausgeführt werden. Hat man sich einen solchen Sound-String zusammengebastelt, wird dessen Anfangsadresse in Adresse an XBIOS(32) übergeben. Wird als Adresse -1 angegeben, so erhält man in Var die Speicheradresse, deren Inhalt gerade gespielt wird.

21.5.3 Interrupt-Sound

Die Struktur dieser XBIOS-Funktion ist nicht gerade einfach zu handhaben und liefert bei nur einem falsch gesetzten Register oder Befehl einen konfusen Klang- und Rauschsalat. Bei richtiger Handhabung können jedoch hiermit ganze Musikstücke mehrstimmig abgespielt werden.

In den String werden mit einem vorangestellten Kommando-Byte versehene Daten-Bytes geschrieben, wobei die Reihenfolge gleichgültig ist, solange immer ein Byte-Paar (eine Ausnahme!) zusammen in den String geschrieben wird und die "Logik" der Kommando-Reihenfolge erhalten bleibt.

Folgende Kommando-Bytes sind gültig:

0 Diesem Kommando folgt ein Byte, das mit seinen 8 Bits das LO-Byte der **Periodendauer** für den 1. Sound-Kanal bestimmt (0 - 255).

1 Das diesem Kommando folgende Byte bestimmt mit seinen unteren 4 Bits das HI-Byte der **Periodendauer** des 1. Sound-Kanals (0 - 15).

2 Wie 0, jedoch für 2. Sound-Kanal.

3 Wie 1, jedoch für 2. Sound-Kanal.

4 Wie 0, jedoch für 3. Sound-Kanal.

5 Wie 1, jedoch für 3. Sound-Kanal.

6 Das hierauf folgende Byte bestimmt mit den unteren 6 Bits die **Rauschgenerator-Frequenz** (0 - 63).

7 Das folgende Byte bestimmt, welche Sound-Kanäle aktiviert werden sollen. Dabei werden die Bit-Werte für die aktiven Kanäle vom Wert 255 abgezogen, z.B.:

&X11111111 (255) = Alle Kanäle aus
&X11111110 (254) = Kanal 1 an
&X11111101 (253) = Kanal 2 an
&X11111011 (251) = Kanal 3 an
&X11110111 (247) = Rauschen für Kanal 1 an
&X11101111 (239) = Rauschen für Kanal 2 an
&X11011111 (223) = Rauschen für Kanal 3 an
&X11000000 (192) = Alle Kanäle mit Rauschen an
&X11011001 (217) = Rauschkanal 3, Sound-Kanal 3 und Sound-Kanal 2 an
etc.

Achten Sie darauf, daß keine Werte kleiner als 192 verwendet werden, da sonst die Floppy-Ports im Sound-Chip angesprochen werden und dies unvorhersehbare Folgen haben kann.

8 Das folgende Byte bestimmt in den unteren 4 Bit die **Lautstärke** für den Sound-Kanal 1 (0 - 15).

9 Wie 8, jedoch für 2. Sound-Kanal.

10 Wie 8, jedoch für 2. Sound-Kanal.

11 Mit folgendem Byte wird die Feinabstimmung für **Hüllkurven-Länge** vorgenommen (Sustain/Hüllkurvenfrequenz 0 - 255).

12 Wie 11, jedoch für Grobabstimmung (0 - 255).

13 Das folgende Byte bestimmt mit dem oberen Nibble die **Hüllkurvenform** (8 - 15):

```
 8  =  \¦\¦\¦\¦\  = Minus-Sägezahn, anfangs fallend
 9  =  \_________ = Anfangs fallend, haltend
10  =  \/\/\/\/\/ = Dreieckwelle, anfangs fallend
11  =  \¦________ = Attack/Cut, anfangs fallend
12  =  /¦/¦/¦/¦/¦ = Plus-Sägezahn, anfangs steigend
13  =  /_________ = Anfangs steigend, haltend
14  =  /\/\/\/\/\ = Dreieckwelle, anfangs steigend
15  =  /¦________ = Attack/Cut, anfangs steigend
```

128 Speichert das folgende Byte (0 - 255) in einem **Zwischenspeicher**.

129 Bildet die einzige Ausnahme bezüglich der Byte-Paare. Diesem Identifikations-Byte müssen immer 4 weitere Bytes folgen. Diese Byte-Folge bewirkt, das der mit 128 zwischengespeicherte Wert in das mit Byte 1 bestimmte Sound-Register geladen wird und dieser Wert nach einer bestimmten Zeitspanne (Byte 4/50) im Register um den Wert von Byte 2 erhöht wird.

In Byte 3 wird ein Endwert bestimmt, bei dessen Erreichen dieser Prozeß abgebrochen wird.

Byte 1 = Betreffendes Sound-Register (0 - 13)
Byte 2 = Schrittweite (0 - 255)
Byte 3 = Endwert (0 - 255)

	Byte 4 = Zeitintervall (0 - 255)
130 bis 255	Alle Werte größer 129 haben dieselbe Kommandowirkung. Das hierauf folgende Byte gibt eine Zeitspanne in 1/50 Sekunden an. Die Routine wird für die angegebene Zeitspanne verzögert bzw. bei Übergabe einer Null beendet. Durch WAVE 0 oder einen einfachen Tastaturklick kann die Routine ebenfalls beendet werden.

Soll der Tastaturklick ausgeschaltet werden, kann dies durch SPOKE &H484,PEEK(&H484) AND NOT 1 erreicht werden. Durch OR 1 statt AND NOT 1 wird der Klick wieder eingeschaltet (siehe Kapitel 22 "Systemvariablen"). Bei der Komplexität dieses Themas wird man als jemand, der es erklären soll, sehr schnell dazu verleitet, etwas falsches oder oberflächliches zu sagen. Gerade zum Thema "Dosound" sind schon viele Halbheiten verbreitet worden.

Es gibt in der Interrupt-Sound-Programmierung soviele "wenns" und "abers", daß man eine umfassende Erklärung aller Umstände nicht im engen Rahmen liefern kann.

Um Ihnen die Möglichkeit zu geben, trotzdem etwas damit zu experimentieren, habe ich ein kleines Programm geschrieben, das Ihnen dabei weitestgehend freie Hand läßt. Das Programm erklärt sich selbst und ist - so glaube ich - so komfortabel, daß damit schon einiges zu erreichen ist. Bei Programmstart wird eine Zahlenreihe von 0 bis 31 dargestellt, unter welcher der jeweilige Inhalt des enstsprechenden Bytes eines 32 Byte langen Strings eingetragen ist. Dieser String ist hier auf 32 Bytes beschränkt, da ich das Programm nicht zu sehr ausweiten wollte. Möchten Sie einen längeren Sound-String bearbeiten, so ließen sich bei entsprechender Änderung des Programms auch mehrere Kolonnen untereinander darstellen. Durch Mausklick auf die Kolonnenwerte kann der Inhalt des gewählten Bytes verändert werden.

Möchten Sie den eingestellten Sound hören, so können Sie den String durch PLAY initialisieren. Er wird dann solange gespielt, bis die Maustaste wieder losgelassen wird. Alles weitere ist aus dem Listing und den Help-Texten zu erkennen.

```
A$=String$(32,0)        ! Puffer-String vorbereiten
Deftext ,0,,6           ! -------------------------.
Text 10,142,"linke Maustaste  = +"                  !
Text 10,150,"rechte Maustaste = -"                  !
Text 10,158,"   ->      = rechts rotieren"          !
Text 10,166,"   <-      = links  rotieren"          !
Text 10,174,"<Help>     = Element kopieren"         !
Text 10,182,"<Delete>   = Element löschen ...       !
                    ... (letztes Element wird Null)"!
Text 10,190,"<Insert>   = Element einfügen ...      !
                    ... (letztes Element fällt raus)"!
```

```
Box 6,130,470,196                                         !
Deftext ,0,,4                                             !
For I%=0 To 31                                            !- Screen-
  Text I%*20+6,80,Str$(I%)                                !  Aufbau
  Box I%*20-1,70,(I%+1)*20-1,110                          !
Next I%                                                   !
Text 10,62,240,"PLAY  LOAD   SAVE   QUIT  ** DOSOUND **"
Line 1,86,638,86                                          !
Line 1,88,638,88                                          !
Line 1,89,638,89                                          !
Line 1,109,638,109                                        !
For I%=0 To 3                                             !
  Box 0,50,39+I%*40,68                                    !
  Box 1,51,38+I%*40,67                                    !
Next I%                                                   !
@Pline                  !--------------------------------'
Do                      ! Endlos-Schleife
  P%=Mousex/20          ! Maus-X-Index holen
  If Mousey>50 And Mousey<68 And Mousek ! Maus auf Menü?
    If P%<2             ! PLAY?
      Void Xbios(32,L:Varptr(A$)) ! Sound-String starten
      Graphmode 3        ! XOR-Modus für Indexbox
      Get 0,90,639,110,Bg$ ! Kolonne als Image speicher
      Repeat             ! Play-Schleife
        Bt%=Xbios(32,L:-1) ! Aktuelles Byte ermitteln
        Box (Bt%-Varptr(A$))*20,90,(Bt%-Varptr(A$))*20+18,108
        Print At(1,1);"Spiele Byte: ";Peek(Bt%) ! Byte anzeigen
      Until Mousek=0     ! Exit, wenn Maustaste = 0
      Put 0,90,Bg$       ! Kolonne restaurieren
      Graphmode 1        ! REPLACE-Modus
      Wave 0             ! Dosound ausschalten
      Print At(1,1);Spc(44)
    Endif
    If P%>1 And P%<4    ! LOAD?
      Fileselect "\*.*","",Fl$  ! Datei wählen
      If Exist(Fl$)      ! Datei existiert?
        Open "I",#1,Fl$ ! Dann öffnen
        Bget #1,Varptr(A$),Min(32,Lof(#1))
        '                ! Max. 32 Bytes in String laden
        Close            ! Datei schließen
        @Pline           ! Neuen Byte-String anzeigen
      Endif
    Endif
    If P%>3 And P%<6    ! SAVE?
      Fileselect "\*.SND",".SND",Fl$  ! Datei wählen
      If Fl$>"" And Fl$<>".SND" And Right$(Fl$)<>"\"
        '                ! Korrekte Dateinamen-Eingabe?
        Bsave Fl$,Varptr(A$),Len(A$) ! Sound-String speichern
      Endif
    Endif
    If P%>5 And P%<8    ! QUIT?
      Stop               1 Dann STOP
      ' Ggfs. im Direktmodus durch CONT Fortsetzung
    Endif
  Endif
  If P%<32 And Mousey<110 And Mousey>90 And Mousek
    ' Mauszeiger auf Kolonnen-Element und Taste gedrückt?
    If Mousek=1          ! Linke Maustaste gedrückt?
      Poke Varptr(A$)+P%,Peek(Varptr(A$)+P%)+1
```

```
    '                  ! Byte-Inhalt um 1 erhöhen
  Else                 ! Rechte Maustaste!
    Poke Varptr(A$)+P%,Peek(Varptr(A$)+P%)-1
  Endif                ! Byte-Inhalt um 1 vermindern
  Text P%*20,100,19,Right$("  "+Str$(Peek(Varptr(A$)+P%)),3)
  '                    ! Neuen Byte-Wert anzeigen
  Pause 4              ! Kleine Klickpause
Endif
Key$=Inkey$            ! Tastatur abfragen
Key%=Asc(Right$(Key$))! Ggfs. ASCII der Taste holen
If Key%                ! <Taste> gedrückt?
  If Key%=77 And Len(Key$)=2 ! <Pfeil-rechts>-Taste?
    Bf$=Right$(A$)     ! Letztes Byte retten
    Bmove Varptr(A$),Varptr(A$)+1,Len(A$)-1 ! Rotiere...
    '                  ! ...String um ein Byte nach rechts
    Poke Varptr(A$),Asc(Bf$) ! Letztes Byte vorn wieder rein
  Endif
  If Key%=75 And Len(Key$)=2 ! <Pfeil-links>-Taste?
    Bf$=Left$(A$)      ! Erstes Byte retten
    Bmove Varptr(A$)+1,Varptr(A$),Len(A$)-1 ! Rotiere...
    '                  ! ...String um ein Byte nach rechts
    Poke Varptr(A$)+Len(A$)-1,Asc(Bf$) ! Erstes Byte hinten..
  Endif                                ! ...wieder reinpoken
  If Key%=82 And Len(Key$)=2 ! <Insert> gedrückt?
    ' Beim Einfügen wird ein Null-Byte in die angeklickte
    ' Kolonnen-Position geschoben und alle rechts davon
    ' liegenden Bytes um eine Position nach rechts
    ' versetzt. Das letzte Byte fällt dabei aus der Kolonne.
    Print At(1,1);"<Insert>-Feld anklicken (rechte Taste=Undo)"
    @Wait              ! Auf Mausklick warten
    Print At(1,1);Spc(44)
    If Mousek=1        ! Linke Maustaste gedrückt?
      P%=Int(Mousex/20) ! X-Index ermitteln
      Bmove Varptr(A$)+P%,Varptr(A$)+P%+1,Len(A$)-(P%+1)
      '                ! Rest-Kolonne um 1 Byte nach rechts
      Poke Varptr(A$)+P%,0 ! Null in freie Position poken
    Endif
  Endif
  If Key%=98 And Len(Key$)=2 ! <Help> gedrückt?
    ' Beim Kopieren eines Bytes wird der Inhalt des
    ' zuerst angeklickten Kolonnen-Bytes in das
    ' als zweites gewählte Kolonnen-Byte geschrieben.
    ' Das Quell-Byte bleibt dabei unverändert.
    Print At(1,1);"Quell-Feld anklicken (rechte Taste=Undo)"
    @Wait              ! Auf Mausklick warten
    If Mousek=1        ! Linke Maustaste gedrückt?
      P1%=Int(Mousex/20) ! Quell-X-Index berechnen
      Print At(1,1);"Ziel-Feld anklicken (rechte Taste=Undo)"
      Pause 10         ! Kleine Klickpause
      @Wait            ! Auf Mausklick warten
      Print At(1,1);Spc(44)
      If Mousek=1      ! Linke Maustaste gedrückt?
        Pause 10       ! Kleine Klickpause
        P2%=Int(Mousex/20) ! Ziel-X-Index berechnen
        Poke Varptr(A$)+P2%,Peek(Varptr(A$)+P1%)
      Endif            ! Quell-Byte ins Ziel-Byte schreiben
    Endif
  Endif
  If Key%=127          ! <Delete> gedrückt?
```

```
        ' Beim Löschen eines Kolonnen-Bytes wird die rechts
        ' von der angeklickten Kolonnen-Position liegende
        ' Rest-Kolonne um ein Byte nach links verschoben. Das
        ' frei werdende letzte Byte wird mit Null überschrieben.
        Print At(1,1);"<Delete>-Feld anklicken (rechte Taste=Undo)"
        @Wait              ! Auf Mausklick warten
        Print At(1,1);Spc(44)
        If Mousek=1        ! Linke Maustaste gedrückt?
          P%=Int(Mousex/24) ! X-Index des Lösch-Bytes holen
          Bmove Varptr(A$)+P%+1,Varptr(A$)+P%,Len(A$)-(P%+1)
          '                ! Kolonnen-Rest um ein Byte nach links
          Poke Varptr(A$)+Len(A$)-1,0 ! Hinterstes Byte löschen
        Endif
      Endif
      @Pline               ! Geänderte Byte-Kolonne darstellen
      Pause 10             ! Kleine Klickpause
    Endif
  Loop
  Procedure Pline          ! Erledigt das Schreiben der Kolonne
    For I%=0 To 31         ! 32 Positionen
      Text I%*20,100,19,Right$("  "+Str$(Peek(Varptr(A$)+I%)),3)
    Next I%                ! Aktuellen Byte-Inhalt holen und...
  Return                   !                   ...ausgeben
  Procedure Wait           ! Wartet auf einen Mausklick
    Repeat
    Until Mousek
  Return
```

Var=XBIOS(33,Modus) *-SETPRT-*

Es kann die aktuelle Druckereinstellung gelesen, bzw. eine neue gesetzt werden. Wird für Modus -1 übergeben, erhält man in Var einen Bit-Vektor, der die aktuelle Einstellung beschreibt. Ein von -1 ungleicher Wert wird als neue Einstellung übernommen.

Var Modus	gesetzt	nicht gesetzt
Bit 0	Matrixdrucker	Typenraddrucker
Bit 1	Colordrucker	Schwarz/Weiß-Drucker
Bit 2	Atari	Epson
Bit 3	Test	Maximum
Bit 4	Centronics	RS232
Bit 5	Endlos	Einzelblatt

```
Standard-Einstellung für Epson: XBIOS(33,6)
```

Var=XBIOS(34,Modus) *-KBDVBASE-*

Liefert in Var die Startadresse einer Tabelle. In dieser Tabelle stehen der Reihe nach sieben Startadressen (Long) von Maschinen-Routinen, die Aufgaben für den IKB (Tastaturprozessor) erledigen.

Aufbau der Tabelle:

Var+0	**MIDI-Eingabe-Routine (midivec): Adresse einer Routine, die nach Eingang von Daten am MIDI-In-Port (Byte liegt in D0) dieses Byte in den MIDI-Puffer schreibt.**
Var+4	**Tastatur-Fehler-Behandlung (vkbderr): Zeigt auf eine Routine zur Behandlung von Tastatur-Überlauf (Meldung durch IKB-ACIA).**
Var+8	**MIDI-Fehler-Behandlung (vmiderr): Zeigt auf eine Routine zur Behandlung von MIDI-Überlauf (Meldung durch MIDI-ACIA).**
Var+12	**Tastatur-Status-Paket (statvec): Zeigt auf eine Routine, die die vom Tastatur-ACIA gelieferten Tastatur-Datenpakete verarbeitet.**
Var+16	**Maus-Status-Paket (mousevec): Zeigt auf eine Routine, die die vom Tastatur-ACIA gelieferten Maus-Datenpakete verarbeitet.**
Var+20	**Zeit-Paket (clockvec): Zeigt auf eine Routine, die die vom Tastatur-ACIA gelieferten Zeit-Datenpakete verarbeitet.**
Var+24	**Joystick-Status-Paket (joyvec): Zeigt auf eine Routine, die die vom Tastatur-ACIA gelieferten Joystick-Datenpakete verarbeitet.**

Sobald ein Interrupt durch ein entsprechendes Ereignis ausgelöst wurde, enthält das CPU-Register A0 bei den letzten vier Routinen die Anfangsadresse des zu verarbeitenden Datenpakets. Falls hier eigene Routinen eingesetzt werden sollen, dürfen Sie nicht mehr als 1/1000 Sekunde benötigen und müssen mit RTS abgeschlossen sein.

Var=XBIOS(35,Delay,Repeat) *-KBRATE-*

Liefert bzw. bestimmt die Reaktions-Verzögerung und die Wiederholungsrate der Tastatur. Damit ist das gemeint, was man auch mit den beiden Schiebereglern (Tastenfinger, bzw. Hase und Schildkröte) im Panel des Control-Accessories einstellen kann. Die Reaktionsverzögerung (Delay) ist die Zeitspanne, die zwischen dem ersten Druck auf eine Taste und dem Reagieren des Cursors vergeht. Mit der Wiederholungsrate (Repeat) ist die Zeitspanne gemeint, die der Cursor nach der ersten Reaktion benötigt, um bei gedrückt gehaltener Taste jeweils um eine Cursor-Position weiterzuspringen.

In Delay wird entweder eine -1 (keine Änderung) oder die neue Verzögerungsrate erwartet, während in Repeat der Wert -1 (keine Änderung) oder die neue Taktfrequenz übergeben wird. Die Normaleinstellung ist Delay = 10 und Repeat = 3. In Var wird ein Longword zurückgeliefert, wovon das HI-Byte des LO-Words die alte Reaktionsverzögerung (Delay) und das LO-Byte des LO-Words die alte Taktfrequenz (Repeat) darstellt.

Der Wert 0 für Delay, bzw. 1 für Repeat steht jeweils für schnell. Je größer der Wert, umso langsamer. Wird in Repeat eine Null übergeben, so ist der Tastatur-Repeat ausgeschaltet.

Var=XBIOS(36,L:Adresse) *-PRTBLK-*

Dies ist eine erweiterte Hardcopy-Funktion. Während XBIOS(20 SCRDMP nur den Ausdruck einer Standard-Hardcopy zuläßt, können hier verschiedene Parameter übergeben werden.

Dies Parameter sind in einem Block zusammenzufassen, dessen Startadresse dann in Adresse an die Funktion übergeben wird.

Blockaufbau:

1 Long	Bildschirm-Adresse.
1 Word	??.
1 Word	Bildschirmbreite.
1 Word	Bildschirmhöhe.
1 Word	??.
1 Word	??.
1 Word	Bildschirm-Auflösung (siehe XBIOS(4) GETREZ).
1 Word	Drucker-Auflösung (0 oder 1).
1 Long	Adresse der Farb-Palette.
1 Word	Druckertyp (0, 1, 2 oder 3).
1 Word	Drucker-Port (0 = Centronics/1 = RS232).
1 Long	Adresse der "Halbtonmaske" (??).

Var=XBIOS(37) *-VSYNC-*

Diese Funktion ist mit dem BASIC-VSYNC-Befehl identisch (siehe dort).

Var=XBIOS(38,L:Adresse) *-SUPEXEC-*

Es kann die Startadresse einer Maschinenroutine übergeben werden. Diese Routine wird dann im Supervisor-Modus ausgeführt. Die Umschaltung durch GEMDOS(32) Super ist hier nicht nötig.

Var=XBIOS(39) *-PUNTAES-*

Befindet sich das AES im RAM, kann es mit dieser Funktion abgeschaltet werden. Das Ergebnis ist ein Reset. Bei ROM-TOS-Maschinen konnte ich keine Wirkung feststellen.

22. Systemvariablen

&H424 1 Word *-memctrl-*
Kopie des Konfigurationswertes im Memory-Controller.

&H426 1 Long *-resvalid-*
Wird hier der Wert &H31415926 eingetragen, kann damit erreicht werden, daß bei einem Reset über den Reset-Vektor in Routinen gesprungen wird, die auf den Reset reagieren.

Ohne genaueste Kenntnis des internen Vorgangs ist hiermit leider nichts zu erreichen.

&H42A 1 Long *-resvector-*
Dieses ist der eben genannte Reset-Vektor, der allerdings keinen Inhalt aufweist. Soll auf den Reset reagiert werden,wird hier die Adresse der reagierenden Routine eingetragen.

&H42E 1 Long *-phystop-*
In dieser Adresse ist das Ende des physikalischen RAMs eingetragen. Normalerweise ist dies 767 Bytes hinter dem Bildschirmspeicher.

&H432 1 Long *-membot-*
Dies ist die Anfangsadresse des Benutzerspeichers oberhalb des vom Betriebssystem verwendeten Speichers am RAM-Anfang.

&H436 1 Long *-memtop-*
Endadresse des Benutzerspeichers. Normalerweise ist dies der Beginn des Bildschirmspeichers.

&H440 1 Word *-seekrate-*
Ist Ihnen Ihre Floppy zu langsam? Wenn Sie in diese Adresse den Wert 2 Lpoken, wird sie etwas schneller.

Damit wird die Geschwindigkeit der Schreib-/Lesekopf-Bewegung von einem Track zum nächsten bestimmt.

0 6 Millisek./1 = 12 Millisek.
2 2 Millisek./3 = 3 Millisek. Der Default-Wert ist 3

&H442 1 Word *-timer_ms-*

Diese Adresse enthält die Differenz zwischen zwei System-Timer-Aufrufen in Millisekunden (normal: 20).

&H444 1 Word *-fverify-*

Diese Speicherstelle bestimmt, ob bei Floppy-Schreibzugriffen automatisch ein Verify ausgeführt wird. Wenn Sie das Verify unterdrücken möchten (Floppy schreibt schneller), müssen Sie diese Adresse auf Null setzen. Jeder andere Wert schaltet das Verify wieder ein.

&H446 1 Word *-bootdev-*

Enthält die Nummer des Laufwerks, von dem das System gebootet wurde. Bei ROM-TOS-Maschinen ist das relativ uninteressant.

```
0 = A:  /  1 = B:
```

&H448 1 Word *-palmode-*

Liefert Auskunft, ob das Sytem im 50-Hertz-PAL-Modus (<>0) oder im 60-Hertz-NTSC-Modus (= 0) arbeitet. Veränderungen an dieser Adresse haben keinen Sinn, da der Modus nur während des Bootens installiert werden kann.

&H44A 1 Word *-defshiftmod-*

Bei Umschaltung vom Monochrom- in den Farbmodus liefert dieses Adresse dem System die gewünschte Farb-Auflösung. Ohne System-Kenntnisse kann dieses Adresse nur gelesen werden.

```
0 = Lowres/1 = Midres
```

&H44C 1 Word *-sshiftmd-*

Auch diese Adresse kann nur gelesen werden. Sie liefert den gleichen Wert, der auch durch XBIOS(4) erfahren werden kann.

```
0 = Lowres/1 = Midres/2 = Hires
```

&H44E 1 Long *-v_bas_ad-*

Enthält einen Zeiger auf die Startadresse des logischen Bildschirms (siehe XBIOS(3)). Wollen Sie die Screen-Adressen verändern, können Sie dies am besten mit XBIOS(5) tun.

&H452 1 Word *-vblsem-*

Durch Setzen einer Null in dieser Adresse können die in &H454 eingetragenen Vertikal-Blanc-Interruptroutinen gesperrt werden. Dadurch wird dann z.B. auch die Abbruchfunktion des GFA-BASIC außer Kraft gesetzt. Durch Eintragen einer 1 werden die Routinen wieder aktiviert.

&H454 1 Word *-nvbls-*

In diesem Longword steht die Anzahl der auszuführenden Vertikal-Blank-Interruptroutinen. Als Defaultwert ist hier eine 8 (8 Routinen) eingetragen. Vom System wird nur eine VBL-Routine installiert. Die anderen 7 Routinen können frei bestimmt werden. Durch Heraufsetzen der Zahl können entsprechend viele Routinen angemeldet werden (siehe Prozedur Slow unter BYTE{}/CARD{}/LONG{}).

&H456 1 Long *-vblqueue-*

Dies ist ein Zeiger auf eine Liste von Adressen der in &H454 angemeldeten VBL-Routinen. Im Default-Zustand ist diese Liste 8 Longwords lang (8 eingetragene Routinen). Durch Lpeek(Lpeek(&H456)) erfahren Sie die Startadresse der ST-Standard-Mausroutine. Das zweite Longword Lpeek(Lpeek(&456)+4) zeigt sehr wahrscheinlich auf eine Adresse im BASIC-Interpreter, wo sich eine vom BASIC eingetragene Routine befindet. Sollte das nicht der Fall sein, wurden vorher durch ein Auto-Boot-Programm eine oder mehrere VBL-Routinen angemeldet und der Eintrag der BASIC-Routine verschiebt sich um dementsprechend viele Longwords in der Liste.

Die nächsten Einträge hinter dem BASIC-Eintrag müßten dann Null sein. Dort kann man dann die Adressen seiner eigenen Interrupt-Routinen eintragen und damit erreichen, daß diese Routinen dann vom ST treu und brav bei jedem VBL-Interrupt ausgeführt werden. Sollen mehr als 8 VBL-Routinen ausgeführt werden, ist die Liste entsprechend zu erweiteren (siehe Prozedur Slow unter BYTE{}/CARD{}/LONG{}), in einen ausreichend großen Speicherbereich zu verlegen, in &H456 die neue Tabellen-Adresse und in &H454

die neue Anzahl einzutragen. Ohne fundierte Assembler-Kenntnisse ist hier jedoch nicht viel auszurichten (außer Slow!).

Übrigens können Sie die Abbruchfunktion des BASIC total ausschalten, indem Sie in das vierte Byte der V2.xx- BASIC-Interruptroutine einfach eine Null schreiben. Wollen Sie in V2.xx auch noch den Direktmodus im Interpreter ausschalten, schreiben Sie in das fünfte Byte einfach ebenfalls eine Null. Nur darf danach kein On Break Cont-Befehl mehr ausgeführt werden, da sonst die Break-Umschaltung im Interpreter durcheinander gerät. Wollen Sie beides wieder einschalten, poken Sie in das vierte Byte eine 27 und in das fünfte Byte eine 2. Die BASIC-Interrupt-Routine kann man in der Liste dadurch identifizieren, daß sie ca. 26 KByte über der BASIC-Basepage, also innerhalb des Interpreters liegen muß.

&H45A 1 Long *-colorptr-*
Hier kann die Adresse einer 16 Words umfassenden Farbpalette eingetragen werden. Diese Palette wird dann beim nächsten VBL-Interrupt geladen, sofern der VBL-Interrupt nicht gesperrt ist. Damit keine Palette geladen wird, ist der Wert 0 einzutragen.

&H45E 1 Long *- screenptr-*
Diese Adresse wird als Zeiger auf die Startadresse des logischen Bildschirmspeichers (vgl. XBIOS(3) und XBIOS(5)) interpretiert. Das neue Video-RAM wird dann beim nächsten VBL-Interrupt installiert. Soll nicht ständig das Video-RAM installiert werden, muß diese Adresse Null enthalten.

&H462 1 Long *-vbclock-*
In diesem Longword werden die seit Systemstart erzeugten Interrupts gezählt.

&H466 1 Long *-frclock-*
Hier werden die seit Systemstart ausgeführten Interrupt-Routinen gezählt.

&H46A 1 Long *-hdv_init-*
Zeiger auf eine Sprungtabelle für die Initialisierung einer Hard-Disk.

&H46E 1 Long *-swv_vec-*

Sprungvektor bei Änderung der Bildschirmauflösung im Farbbetrieb (siehe 3. Parameter bei XBIOS(5)).

&H472 1 Long *-hdv_bpb-*

Sprungvektor zur Ermittlung des Bios-Parameter-Blocks für Hard-Disks.

&H476 1 Long *-hdv_rw-*

Zeiger auf eine Routine, die das Lesen und Schreiben von/auf Hard-Disks erledigt.

&H47A 1 Long *-hdv_boot-*

Zeiger auf eine Routine, die ausgeführt wird, wenn von einer Hard-Disk aus das System gebootet werden soll.

&H47E 1 Long *-hdv_mediach-*

Zeiger auf eine Routine, die bei Hard-Disks die Media-Change-Kontrolle durchführt.

&H484 1 Byte *-conterm-*

4-Bit-Vektor:

		Gesetzt	Nicht gesetzt
Bit 0 ->	Tastatur-Klick	Ein	Aus
Bit 1 ->	Tastatur-Repeat	Ein	Aus
Bit 2 ->	Chr$(7)-Klingel	Ein	Aus
Bit 3 ->	HI-Byte von BIOS(2) Liefert Status der Umschalttasten	Ja	Nein

&H486 1 Long *-trp14ret-*

Rücksprungadresse nach TRAP#14-Aufruf (XBIOS-Trap)

&H48E 4 Longs *-themd-*

Mit der BIOS-Funktion 0 wird die Adresse des Memory-Parameter-Blocks ermittelt. Als Rückgabewert erhält man die Adresse dieses Memory Descriptors. Weiteres siehe unter BIOS(0).

&H4A2 1 Long *-savptr-*

Eine Möglichkeit, sich Einblick in den Stand der Prozessor-Register zu verschaffen, hat man hierdurch. Bei jedem BIOS-Aufruf werden die Register-Inhalte in einem bestimmten Bereich gesichert. Der Inhalt dieser Adresse stellt einen Zeiger auf die Anfangsadresse dieses Bereichs dar.

&H4A6 1 Word *-nflops-*

Diese Adresse liefert einen Wert, der die Anzahl der angeschlossenen Laufwerke angibt. Wie bei BIOS(10) wird auch hier immer davon ausgegangen, daß zwei Laufwerke (A und B) mindestens angeschlossen sind.

&H4AC 1 Word *-save_row-*

Dieses Word dient als Puffer für die Cursor-Positionierung.

&H4AE 1 Long *-save_context-*

Zeigt auf einen Puffer für die Exception-Verarbeitung.

&H4BA 1 Long *-hz_200-*

Dies ist der System-Timer, der vom BASIC-Befehl TIMER benutzt wird (vergl. Print Timer'"Lpeek(&H4BA)).

&H4C2 1 Long *-drvbits-*

Aus dieser Adresse kann ermittelt werden, welche Laufwerke zur Zeit angeschlossen sind. Es handelt sich hier um einen Bit-Vektor. Ist Bit 0 gesetzt, heißt das, daß Floppy A angeschlossen ist. Bit 1 steht für Floppy B, Bit 2 für C usw. Auch hier geht das System davon aus, daß generell mindesten Drive A und B angeschlossen sind (siehe XBIOS(10)).

&H4C6 1 Long *-dskbufp-*

Um nicht bei jedem Diskettenzugriff die aktuellen Disketten-Attribute von Disk holen zu müssen, richtet sich das System einen Disk-Buffer ein, in welchem der aktuelle Boot-Sektor abgelegt wird. Dieser 1024 Byte große Buffer beginnt bei der hier eingetragenen Adresse.

&H4CE 8 Longs *-vbl_list-*

Dies ist die VBL-Routinenliste, deren Adresse durch Lpeek(&H456) erfragt werden kann. In ihr sind der Reihe nach alle VBL-Routinen eingetragen, welche bei jedem VBL-Interrupt ausgeführt werden sollen. Wie oben bereits erwähnt, können auch mehr als acht Routinen angemeldet werden, wenn dieser entsprechend verlängerte Vektor an eine ausreichend große Speicherstelle verlegt wird, die neue Anzahl in &H454 und der neue Zeiger in &H456 eingetragen wird (siehe Prozedur Slow unter BYTE{}/CARD{}/LONG{}).

&H4EE 1 Word *-dumpflg-*

Eine 1 in dieser Adresse zeigt an, daß gerade eine Hardcopy gefertigt wird (siehe unter HARDCOPY).

&H4F0 1 Word *-prtabt-*

Dies ist das Printer-Time-Out-Abort-Flag. Wozu dieses System-Flag gut ist, konnte ich nicht in Erfahrung bringen, der Default-Wert ist 252. Ich nehme an, daß dieses Flag dann manipuliert wird, wenn bei einer Druckerausgabe der Drucker nicht empfangsbereit ist und nach ca. 30 Sekunden ein Printer-Time-Out gemeldet wird.

&H4F2 1 Long *-sysbase-*

Diese Adresse zeigt auf die Startadresse des Betriebssystems.

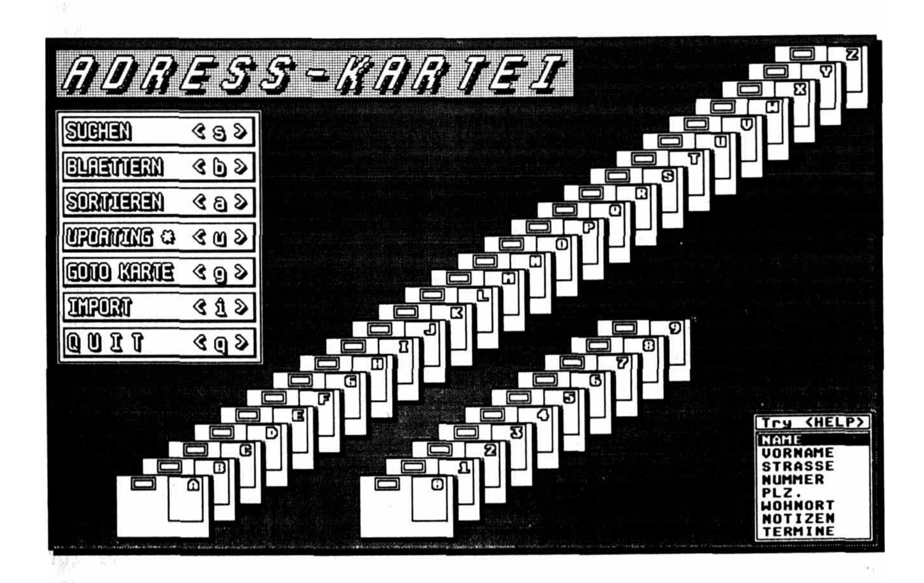
&H4FA 1 Long *-end_os-*

Hier ist das Ende des Betriebssystems im RAM eingetragen. Das Betriebssystem benötigt auch bei einem ROM-TOS einen gewissen Bereich des RAM für sich. Das Ende dieses RAM-Bereichs ist hier gemeint.

&H4FE 1 Long *-exec_os-*

Dies ist ein Zeiger auf die Startadresse des AES innerhalb des Betriebssystems.

23. Vario-RAM-Kartei RAMKART



"Das obligatorische Beispiel-Programm", werden Sie jetzt vielleicht sagen, womit Sie auch recht haben. In diesem Fall habe ich mich jedoch bemüht, ein Programm zu konstruieren, das mehrere nützliche Aspekte gleichzeitig erfüllt. Es ist:

- vernünftig einsetzbar und nutzbar
- sehr lehrreich
- noch überschaubar
- streng strukturiert (obwohl nicht 100%ig optimiert)
- äußerst komfortabel und vielseitig
- schnell
- und zweckmäßig

Es ist wieder ein Datei-Verwaltungsprogramm. Durch einige Eigenschaften tanzt es dabei jedoch gewaltig aus der üblichen Reihe. "Normale" BASIC-Dateiverwaltungen benutzen weitestgehend (oder auch ausschließlich) Standard-Diskettenbefehle wie PRINT#, INPUT# oder GET#, PUT#. Dies hat seine Vorteile insofern, als man sich generell "innerhalb" der BASIC-üblichen Struktur hält.

Genau in diesem Punkt hebt sich jedoch RAMKART von der Masse ab. Es benutzt eine eigene Speicher- und Disketten-Verwaltung und

bietet dadurch eine wesentlich platzsparendere und schnellere Arbeitsweise. Am meisten ist dies an der Geschwindigkeit zu erkennen. Da RAMKART ausschließlich im RAM arbeitet (deshalb RAMKART) und nur wenn es ausdrücklich gewünscht wird, auch die Diskettendaten aktualisiert, kann es Befehle wie z.B. BMOVE, BGET# und BPUT# verwenden, um die Daten blockweise zu transportieren. Es kann sein, daß ISAM- Dateien (Index-Sequencial-Aces-sMemory) oder matrix-indizierte Begriffs-Register (wie z.B. die assoziative THEMADAT-Datei) in mancher Hinsicht schneller sind. Sie sind dafür jedoch nicht so einfach zu begreifen wie RAMKART.

Wer das Demprogramm Vario-RAM-Kartei aus den ersten Auflagen kennt, kann dieses getrost vergessen. In manchen Punkten ähneln sich die beiden Programme und sehen grafisch auch ähnlich aus, nur - zu tun haben sie nicht mehr viel miteinander.

Ich gehe nun im folgenden davon aus, daß Sie kein völlig "mundgerechtes" Programm zu sehen bekommen wollen, sondern daß Sie sich intensiver mit der Programmierung und dem Charakter des Programms auseinandersetzen wollen. Es liegt mir also nicht viel daran, ein fertiges, "kommerzielles" Programm zu präsentieren, als vielmehr Sie zur "Mitarbeit" bei der Entwicklung anzuregen. Nichts desto weniger braucht sich RAMKART hinter den meisten kommerziellen und professionellen Programmen nicht zu verstecken. Aber das werden Sie schon noch feststellen.

Um jenen, die dieses Programm raubkopieren möchten, ein kleines Bein zu stellen, sind einige Optionen aus dem Programmlauf nicht direkt zu erkennen, sondern ergeben sich aus der hiesigen Beschreibung und dem Programmkommentar. Außerdem steckt im Sortier-Algorithmus ein winzig kleiner Fehler, der nur zu beheben ist, wenn man sich näher mit dem Listing auseinandersetzt. Dieser Fehler besteht darin, daß in der Sortierfolge bei jedem Sortierlauf eine Karte versetzt wird und die unterste Karte nur mit dem ASCII-Zeichen 255 (Hochstrich) gefüllt ist. Dieser Mangel tut dem Nutzen des Programms keinen Abbruch, solange man die jeweils letzte Karte nicht mit wichtigen Daten belegt.

Stört Sie diese eine "verunstaltete" Karte, so können Sie sie einfach dadurch entfernen, indem Sie die Karte "aufschlagen" und mit <Delete> aus dem Speicher entfernen.

Soviel zur Vorrede. Nun folgt eine - aus oben genanntem Grund - etwas oberflächliche Bedienungsanleitung.

Internes

Wie gesagt, beinhaltet RAMKART eine eigene Speicherverwaltung. Der BASIC-Speicher wird durch RESERVE auf das nötigste beschränkt. Die Datenblöcke (Karten) werden dann hintereinander ab HIMEM+20000 abgelegt. Die 20000 Bytes gelten hier eigentlich nur für die V2.02-Version, da diese direkt hinter HIMEM das aktuelle Disk-Direktory ablegt (warum, weiß nur Frank Ostrowski allein). Für die Versionen V2.0 und V3.0 ist dieses Offset nicht nötig. In der entsprechenden Programmzeile können Sie dann also +20000 streichen.

Als Transfer-Befehl gilt hauptsächlich BMOVE, bzw. sofern die Disk-Daten aktualisiert werden sollen, die Befehle BGET# und BPUT#. Nur zur Verarbeitung der einzelnen Karten wird die jeweilige Karte in einen String-Puffer übertragen, der dann mit entsprechenden String-Befehlen (INSTR(), LEFT$(), MID$() etc.) weiterverarbeitet wird.

Dieses Verfahren erfordert einerseits ein äußerst sorgfältige Programmierung und penible Vorsicht im Umgang mit den jeweiligen Block-Adressen, bietet jedoch als Ausgleich eine wesentlich freiere Programmgestaltung, sowie die Möglichkeit, das Programm schneller und kürzer zu machen. Ein vergleichbares Programm, daß auschließlich mit Feldvariablen und Standard-BASIC arbeitet, wäre mit Sicherheit langsamer und länger, sowie komplizierter in der Programmierung.

Das Programm benötigt zu seinem Betrieb eine Initialisierungsdatei. Bei Programmstart werden Sie aufgefordert, zwischen der Datei-Bearbeitung und der Datei-Erstellung zu wählen. Auf der beiliegenden Diskette finden Sie schon eine Mini-Demodatei mit dem Namen ADRESSEN.DAT. Mit dem gleichen Namen, jedoch mit der Extension .DEF finden Sie außerdem noch die erwähnte Initialisierungs-Datei. In dieser Datei sind sämtliche für den Betrieb der jeweiligen Datei wichtigen internen Daten abgelegt. Wählen Sie bei Programmstart die Option DEFINE, so werden Sie aufgefordert, der Reihe nach die Titel für die einzelnen Kartenzeilen, sowie die Längen der jeweils zuzuordnenden Einträge anzugeben. Nachdem Sie das getan haben, geben Sie noch die Anzahl der Karten an, die Ihre Datei haben soll und abschließend den gewünschten Disk-Dateinamen, sowie die gewünschte Bildschirm-Überschrift. Die Datei ist nun definiert und die entsprechende .DEF-Datei wird auf der Diskette abgelegt. Nachdem das erledigt ist, werden Sie nochmals gefragt, ob Sie eine Datei bearbeiten oder entwerfen möchten. Durch OPEN erscheint eine FILESELECT-Box, aus der Sie nun die gewünschte .DEF-Datei auswählen können.

Beim ersten Aufruf einer Kartei wird zuerst ein in der Größe schon kompletter Karteikasten auf die aktuelle Diskette geschrieben. Dieser gilt von nun an als Disketten-Puffer für sämtliche Karteikarten. Bei der Größe der Datei sind Sie allein von dem auf der Diskette und im RAM freien Speicherplatz beschränkt. Übrigens werden beim Laden der Datei **alle** Karten "en bloc" in den Speicher geladen. Bei größeren Dateien nimmt dies natürlich einige Zeit in Anspruch. Da Sie dann jedoch - vorausgesetzt, das UPDATING ist aus - ausschließlich im RAM arbeiten, gleicht die Arbeitsgeschwindigkeit diesen Zeitaufwand bei weitem wieder aus.

Bedienung

Sie haben nun vielfältige Möglichkeiten, die aktuelle Datei zu bearbeiten. Auf der linken Seite finden Sie ein Menü:

Suchen
Es erscheint eine Eingabe-Box, in welcher der erste Suchbegriff eingegeben wird. Anschließend kann ein weiterer Begriff eingegeben werden, der zusammen mit dem ersten im ODER- oder UND-Modus gesucht wird. Dabei wird immer nur die Kartenzeile durchsucht, die rechts unten in der Zeilen-Tabelle angeklickt ist. Ist die letzte Zeile in der Tabelle angeklickt, so wird die komplette Datei durchsucht. Die ODER-Suche zeigt immer dann eine Karte an, wenn entweder der erste oder der zweite Begriff gefunden wurde. Bei der UND-Suche müssen beide Begriffe zugleich in der Zeile, bzw. in der Karte auftauchen, um gefunden zu werden. Wird im Suchmodus eine Karte angezeigt, so kann die Suche durch die linke Maustaste oder einer beliebigen A-Z-Taste bzw. <Space> fortgesetzt werden. Druck auf die rechte Maustaste oder <Esc> bricht die Suche ab.

Blättern
Die erste Karteikarte wird angezeigt. Es kann nun durch <Return> vorwärts oder <Control><Return> rückwärts geblättert werden. <Esc> kehrt zum Kartei-Desktop zurück.

Sortieren
Die Kartei wird komplett sortiert. Dabei kann durch die Zeilen-Tabelle rechts im Bild angegeben werden, von welcher Zeile der Anfangsbuchstabe als Kriterium gewählt werden soll. Die Tabellenzeile

kann durch Mausklick oder die beiden Vertikal-<Pfeiltasten> gewählt werden.

Updating
Dies ist eine Umschaltfunktion, die darüber entscheidet, ob nach Änderungen einer Karte oder nach dem Sortieren die Disketten-Kartei ebenfalls entsprechend geändert werden soll. Ist das UPDATING ausgeschaltet, so wird ggfs. bei Programmende gefragt, ob die jetzt aktuelle Kartei auf die Diskette zurückgeschrieben werden soll. Bei Programmstart ist das UPDATING eingeschaltet.

Goto Karte
Es erscheint eine Eingabebox, die die Eingabe einer Kartennummer erwartet. Anschließend wird die gewählte Karteikarte "aufgeschlagen".

Import
Die Kartei kann ab Karteistart mit einer beliebigen Disk-Datei überschrieben werden. Ist die Datei kürzer als die Karteilänge, wird ohne Fehler die Disk-Datei in Ihrer Länge übertragen. Ist die Disk-Datei größer, wird Sie ohne Fehler bei Erreichen der Karteilänge abgeschnitten. Dieselbe Möglichkeit hat man auch für jede einzelne Karte.

Quit
Beendet das Programm. Neben den Menü-Einträgen ist jeweils der Tastatur-Buchstabe angegeben, durch den die gewünschte Option ebenfalls ausgelöst werden kann.

Weiterhin finden Sie in der Bildmitte zwei diagonale Register-Ketten, und zwar von A bis Z und von 0 - 9. Jeder dieser Register-"Reiter" steht für einen Buchstaben oder eine Ziffer. Wird ein Register-Reiter angeklickt, so wird die Kartei ab Anfang auf Zeilen untersucht, die auf der ersten Zeilenposition das entsprechende Zeichen, bzw. die entsprechende Ziffer enthält. Welche Zeile der Karten dabei ausschlaggebend sein soll, wird auch hier durch Wahl der Zeile in der rechten Zeilen-Tabelle bestimmt. Wird eine solche Zeile gefunden, wird die entsprechende Karte angezeigt. Fortgesetzt oder abgebrochen wird die Suche genauso wie bei SUCHEN. Statt eines Klicks auf den gewünschten Reiter kann bei Buchstaben auch <Shift><A> bis <Shift>-<Z> und bei den Ziffern <0> bis <9> als Taste gedrückt werden.

Die <Help>-Taste hat eine Sonderfunktion. Wird Sie im Kartei-Desktop gedrückt, so wird nach der Angabe einer Kartennummer gefragt. Ab dieser Karte werden dann anschließend all die Zeilen der Karten angezeigt, die - wiederum - rechts in der Zeilentabelle gewählt wurde. Dadurch kann man sich also eine tabellarische Übersicht über alle gleichwertigen Karteneinträge verschaffen.

Ist eine Karte angezeigt, hat man auf folgende Weise die Möglichkeit, zur Kartenänderung "in die Karte" zu gehen.

- <Shift> drücken.

Der Cursor wird auf den Anfang der obersten Zeile gesetzt.

- Mausklick auf die gewünschte Zeile mit linker Taste:

Cursor wird auf den Anfang der Zeile unter dem Mauszeiger gesetzt.

- Mausklick mit rechter Taste:

Cursor wird auf den Anfang der Zeile unter dem Mauszeiger gesetzt und die Zeile gelöscht.

- <Space> wirkt wie Mausklick links.
- <Alternate> wirkt wie Mausklick rechts.

Innerhalb der Karte kann der Cursor durch <Return> nach unten und durch <Control><Return> nach oben bewegt werden. Verlassen wird die Karte durch

- <Alternate>

Es wird anschließend gefragt, ob die Änderungen bestehen bleiben sollen. Wenn nicht, wird der Karten-Inhalt auf den vor Öffnung gültigen Stand restauriert.

- <Shift><Return>

Die Karte wird mit Änderungen in den Speicher übernommen. Dasselbe geschieht, wenn das erste oder letzte Zeichen einer Zeile entweder <Space> oder <Esc> ist und die Zeile dann mit <Return> verlassen wird.Es gibt zwar noch einige komfortable Kleinigkeiten im Programm, aber da Sie sich ja mit dem Listing auseinandersetzen werden, werden Ihnen diese sicher noch auffallen. Wie Sie sehen werden, habe ich mir wie auch im übrigen Buch - große Mühe mit der Programmkommentierung gemacht, so daß ein echter Lerneffekt gegeben ist. Programme dieser Größe sind mit Ihrer Größe vor allem deshalb

lehrreich, da aus Ihnen die komplexen Zusammenhänge eines ausgereiften Programms zu erfassen sind. Als Anfänger wird Sie diese Komplexität vielleicht zuerst abschrecken, aber ich bin der Meinung, daß man sich gutes Programmieren nur anhand guter Lehr-Beispiele aneignen kann. Unbescheidener Weise gehe ich hier davon aus, daß RAMKART bei Ihnen als "gutes Lehr-Beispiel" anerkannt wird. Auch hier bestätigen natürlich Ausnahmen die Regel.

Am Programmende finden Sie als besondere Belohnung noch einige nützliche, allgemein gehaltene Prozeduren, von den ich hier Dcolor besonders hervorheben möchte. Diese Prozedur gibt Ihnen - ähnlich wie das bekannte Control-Accessory - die Möglichkeit, auf sehr komfortable Weise die Farbeinstellung vorzunehmen, und zwar in Midres **und** in Lowres. In Hires ist Dcolor zwar auch einsetzbar, ist aber dort relativ zwecklos, da Schwarz und Weiß trotz aller Editier-Versuche immer Schwarz und Weiß bleiben.

```
*************************************************************************
KEYPAD 0  ! =====> Diese Zeile gilt nur für die V3.0-Version <=====
**************************************************************************
@intro                              ! DEF-Datei wählen und Prog-Inits
@datload                            ! DAT-Datei laden
@main
**************************************************************************
PROCEDURE main
 DO                                 !-----> Programm...  >------.
   @auswahl                         ! Auf Auswahl-Klick warten  |
   @verteiler                       ! Zur Verzweigungs-Prozedur |
   @bildaufbau                      ! Hintergrund restaurieren  |
 LOOP                               !----< ...Hauptschleife  <---'
RETURN
***************************************************************************
PROCEDURE auswahl                   ! Große Auswahl
 regmem%=reg%                       ! Letzten Registerindex merken
 DO                                 !--> Tasten- u. Mausabfrage... >--.
   @desk_taste                      ! Tastaturabfrage                 |
   @auswertung                      ! Auswahl-Analyse                 |
   REPEAT                           ! Warteschleife...                |
   UNTIL MOUSEK=0                   ! ...bis Maustaste=0              |
   EXIT IF wahl%=1 OR wahl%=2 OR dkey%=ASC("q") ! Raus, wenn gewählt  |
 LOOP                               !---< ...bei leerem Desktop  <----'
RETURN
*************************************************************************
PROCEDURE auswertung                ! Auswertung der
 CLR wahl%                          ! Auswahl-Merker klar
 xp%=MOUSEX                         ! Maus-X-Koordinate merken
 yp%=MOUSEY                         ! Maus-Y-Koordinate merken
 mkk%=MOUSEK                        ! Maustasten-Status merken
 IF LEN(dkey$)=2                    ! Sondertaste gedrückt?
   IF dkey%=72                      ! <Pfeil-hoch>-Taste?
     CLR dkey%                      ! Tastenmerker löschen
     xp%=550                        ! Mausposition simulieren
     yp%=(385/t%-(cz%*10/t%))+(zy%*10/t%)-5/t%  !   - '' -
```

```
    ENDIF
    IF dkey%=80                         ! <Pfeil-runter>-Taste?
      CLR dkey%                         ! Tastenmerker löschen
      xp%=550                           ! Mausposition simulieren
      yp%=(385/t%-(cz%*10/t%))+(zy%*10/t%)+15/t%  !   - '' -
    ENDIF
  ENDIF
  IF keyflg%                            ! <Shift><Registertaste> gedrückt?
    IF (dkey%=>65 AND dkey%<=90)        ! A bis Z?
      xp%=75+(dkey%-65)*20              ! Mausposition simulieren
      yp%=16/t%+(25-(dkey%-65))*13/t%!    - '' -      - '' -
    ENDIF
    IF (dkey%=>48 AND dkey%<=57)        ! 0 bis 9 (ohne <Shift>)?
      xp%=260+(dkey%-48)*20             ! Mausposition simulieren
      yp%=224/t%+(9-(dkey%-48))*13/t% !    - '' -      - '' -
    ENDIF
    CLR keyflg%,dkey%                   ! Flag und Taste löschen
  ENDIF
  IF dkey%=0 OR dkey%=13                ! Keine Taste oder <Return>?
    IF xp%>14 AND xp%<164 AND yp%>61/t% AND yp%<250/t%
      '                                 ! Maus auf einem Menüpunkt?
      my%=INT((yp%-62/t%)/27*t%)        ! Menü-Y-Index
      @blinker(0,16,62+my%*27,161,83+my%*27) ! Blinkbox
      menue%=my%+1                      ! Menüindex merken
      wahl%=2                           ! Auswahl-Merker=2
    ENDIF
    IF xp%>540 AND xp%<628 AND yp%>385/t%-cz%*10/t% AND yp%<385/t%
      '                                 ! Maus auf Eintragstitel?
      zy%=INT((yp%-385/t%+cz%*10/t%)/10*t%)  ! Zeilen-Y-Index
      zset%=1                           ! String-Offset zum Kartenanfang
      IF zy%>0                          ! Maus nicht auf 1. Zeile?
        FOR i%=1 TO zy%                 ! String-Offset bis zur gewählten
          ADD zset%,lng%(i%)            ! Zeile addieren
        NEXT i%
      ENDIF
      PUT 538,370/t%-cz%*10/t%,zeilen$ ! Zeilenanzeige restaurieren
      @blinker(1,541,385-cz%*10+zy%*10-2*(t%-1),626,395-cz%*10+zy%*10)
      IF mkk%=2                         ! Rechte Maustaste gedrückt?
        dkey$="  "                      ! Dann <Help>-Taste...
        dkey%=98                        ! ...simulieren
      ENDIF
    ENDIF
    xx1%=INT((xp%-65)/20)               ! Buchstabenregister-X-Index
    ry%=25-INT((yp%-12/t%)/13*t%)       ! Register-Y-Index
    IF xx1%=ry% AND xx1%>-1 AND xx1%<=25 ! Maus auf Buchstaben-Register?
      DPOKE 9954,MAX(20,yp%)            ! Maus ggfs. unter die...
      @blinker(0,67+ry%*20,346-ry%*13,85+ry%*20,337-ry%*13)
      '                                 ! ...GEM-Menüzeile setzen, da
      '                                 ! Sonst Maus blockiert (V2.02)
      reg%=ry%                          ! Registerindex merken
      wahl%=1                           ! Auswahl-Merker=1
    ENDIF
    xx2%=INT((xp%-250)/20)              ! Zahlenregister-X-Index
    IF xx2%=ry% AND xx2%>-1 AND xx2%<=9 ! Maus auf Zahlen-Register?
      @blinker(0,252+ry%*20,346-ry%*13,270+ry%*20,337-ry%*13)
      reg%=ry%+26                       ! Registerindex merken
      wahl%=1                           ! Auswahl-Merker=1
    ENDIF
  ENDIF
```

```
 IF (dkey%=97 OR dkey%=98) AND LEN(dkey$)=1 ! <A> oder <B> gedrückt?
   wahl%=2                          ! Auswahl-Merker=2
   menue%=3-(dkey%-97)              ! Menü-Merker = 3 oder 2
 ENDIF
 IF dkey%=98 AND LEN(dkey$)=2       ! <Help> gedrückt?
   @liste                           ! Dann zur Listing-Routine
   CLR wahl%,menue%                 ! Klar und zurück
 ENDIF
 IF dkey%=ASC("s")                  ! <S> gedrückt?
   wahl%=2                          ! Auswahl-Merker = 2
   menue%=suchen%                   ! Menü-Merker = 1
 ENDIF
 IF dkey%=ASC("i") OR (wahl%=2 AND menue%=import%) ! <I> gedrückt
   '                                ! Oder IMPORT gewählt?
   wahl%=2                          ! Auswahl-Merker=2
   menue%=blaettern%                ! Menü-Merker = 2
   @karte_import(sta%,kges%)        ! Daten importieren
   IF back3%=1                      ! Abbruch geklickt?
     CLR wahl%,menue%               ! Dann klar und zurück
   ENDIF
 ENDIF
 IF dkey%=ASC("g") OR (wahl%=2 AND menue%=springen%) ! <G> gedrückt
   '                                ! Oder GOTO KARTE gewählt?
   wahl%=2                          ! Auswahl-Merker = 2
   menue%=blaettern%                ! Menü-Merker = 2
   @karte_springen                  ! Sprung zur gewünschten Karte
   IF ziel%>0                       ! Sprungziel = Karte 0?
     card%=ziel%-1                  ! Kartenzähler setzen
     sprung%=1                      ! Sprung-Flag setzen
   ELSE                             ! Keine gültige Zielangabe!
     CLR wahl%,menue%               ! Dann klar und zurück
     DEC card%                      ! Kartenzähler rücksetzen
   ENDIF
 ENDIF
 IF dkey%=ASC("u") OR (wahl%=2 AND menue%=update%) ! <U> gedrückt
   '                                ! Oder UPDATING gewählt?
   @automatic                       ! Dann Update-Flag umschalten
 ENDIF
RETURN
*****************************************************************
PROCEDURE verteiler
 IF wahl%=1                         ! Ein Registerreiter wurde gewählt
   @aktion(2)                       ! Dann Action (modus% 2)
 ELSE                               ! Ein Menüpunkt wurde gewählt
   IF dkey%<>ASC("q")               ! <Q>(uit) nicht gedrückt?
     reg%=regmem%                   ! Alten Registerindex merken
     IF menue%=suchen%              ! Menüpunkt "Suchen" gewählt?
       @aktion(0)                   ! Dann Action (modus% 0)
     ENDIF
     IF menue%=blaettern%           ! Menüpunkt "Blättern" gewählt?
       @aktion(1)                   ! Dann Action (modus% 1)
     ENDIF
     IF menue%=sortieren%           ! Menüpunkt "Sortieren" gewählt?
       @sorter                      ! Dann Sortieren
     ENDIF
   ENDIF
   IF menue%=progexit% OR dkey%=ASC("q") ! Menüpunkt "Quit" gewählt
     '                              ! Oder <Q> gedrückt?
     @abbruch                       ! Dann Ende
```

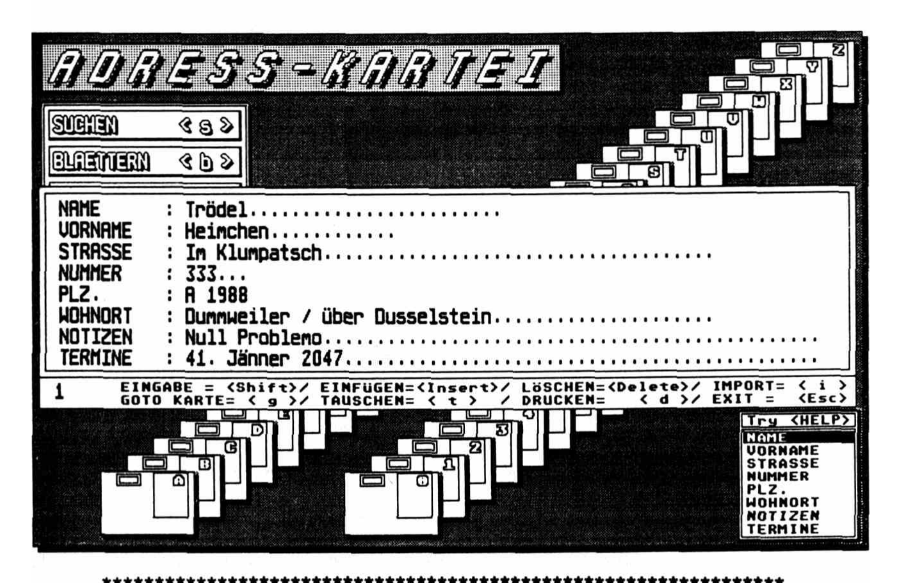
```
    ENDIF
  ENDIF
  CLR dkey%                        ! Tastenmerker löschen
RETURN
********************************************************************
PROCEDURE aktion(modus%)
  IF modus%=0                      ! Suchen?
    CLR such.exit%                 ! Abbruch-Flag klar
    @karte_suchen                  ! Dann suchen
    IF such.exit%=1                ! Sucheingabe abgebrochen?
      GOTO o.ut                    ! Dann zurück
    ENDIF
    PRINT AT(36,23);"BREAK = <Esc>"
    IF zy%=cz%-1                   ! Letzte Eintragszeile "Suchbereich"?
      @blinker(0,540,380-cz%*10+2,626,386)
    ELSE                           ! Suchbereich = Einzelzeile
      @blinker(0,541,385-cz%*10+zy%*10-(t%-1),626,395-cz%*10+zy%*10)
    ENDIF
  ENDIF
  IF sprung%=1                     ! Sprung-Flag gesetzt?
    CLR sprung%                    ! Dann Sprung-Flag klar
  ELSE                             ! Sonst
    CLR card%                      ! Kartenzähler klar
  ENDIF
  DEFMOUSE 2                       ! Biene einschalten
  DO
    nochmal:                       ! Label f. Kartenschleife <-------.
    INC card%                      ! INC Kartenzähler                |
    byte1%=sta%+(card%-1)*sg%      ! Startadresse der Karte          |
    VOID FRE(0)                    ! Garbage Collection              |
    BMOVE byte1%,VARPTR(satz$),sg% ! Karte in Puffer holen           |
    satz.pos%=1                    ! Eintragslängenzähler auf 1      |
    FOR j%=1 TO cz%                ! Alle Einträge durchgehen        |
      ztxt$(j%)=MID$(satz$,satz.pos%,lng%(j%)) ! Eintrag ausschneiden |
      ADD satz.pos%,lng%(j%)       ! Eintragslängenzähler addieren   |
    NEXT j%                        ! Nächster Eintrag                |
    mk%=MOUSEK                     ! Maustaste abfragen              |
    ck%=ASC(INKEY$)                ! Tastatur abfragen               |
    IF modus%=2                    ! Registerreiter gewählt?         |
      IF card%>karten%             ! Hinter letzter Karte?           |
        GOTO o.ut                  ! Dann Anzeige-Ende               |
      ELSE                         ! Noch nicht letzte Karte !       |
        srt$=MID$(satz$,zset%,1)   ! Srt$ = 1. Zeichen der Zeile     |
        IF reg%<26 AND (srt$<>CHR$(reg%+65) AND srt$<>CHR$(reg%+97)) |
          '                        ! Registerbuchstabe <> Srt$?      |
          IF mk%<2 AND ck%<>27     ! Linke oder keine Maustaste bzw. |
            '                      ! <Esc>-Taste nicht gedrückt?     |
            GOTO nochmal           ! Dann nächste Karte  >-----------:
            ' Anm.:                                                  |
            ' Wenn die rechte Maustaste oder die <Esc>-Taste während |
            ' des Suchens gedrückt wird, wird der Suchlauf unter-    |
            ' brochen und die aktuelle Karte angezeigt.              |
          ENDIF                                                      !
        ENDIF                                                        !
        IF reg%>25 AND (srt$<>CHR$(reg%+22)) ! Registerziffer        |
          '                        ! ... ungleich Srt$?              |
          IF mk%<2 AND ck%<>27     ! Linke oder keine Maustaste bzw. |
            '                      ! <Esc>-Taste nicht gedrückt?     |
            GOTO nochmal           ! Dann nächste Karte (s.o.). >----:
```

```
      ENDIF                                                            !
    ENDIF                                                              !
  ENDIF                                                                !
ENDIF                                                                  !
IF modus%=0                     ! "Suchen" gewählt?                    |
  IF card%>karten%              ! Hinter letzter Karte?                |
    GOTO o.ut                   ! Dann Anzeige-Ende                    |
  ELSE                          ! Noch nicht letzte Karte!             |
    CLR s.pos2%,s.pos3%         ! AND/OR-Suchposition klar             |
    IF zy%=cz%-1                ! Letzte Eintragszeile als             |
      '                         ! Suchbereich vorgegeben?              |
      s.pos%=INSTR(satz$,srch$) ! Dann 1. Begriff im gesamten          |
      '                         ! Kartentext suchen                    |
      ' Anm.:                                                          |
      ' Wird im Zeilenanzeigefenster die letzte Zeile gewählt,         |
      ' wird die gesamte Kartei durchsucht. Bei Vorgabe einer          |
      ' anderen Zeile wird nur in der angegebenen Zeile gesucht!       |
    ELSE                        ! Andere Zeile als Suchbereich !       |
      s.pos%=INSTR(MID$(satz$,zset%,lng%(zy%+1)),srch$) ! Nur in       |
      '                         ! ... gewünschter Zeile suchen         |
    ENDIF                                                              !
    IF flg_and%                 ! AND-Modus gewählt?                   |
      IF zy%=cz%-1              ! Suchbereich = letzte Zeile?          |
        s.pos2%=INSTR(satz$,srch2$)! AND-Begriff suchen                |
      ELSE                      ! Andere Zeile als Suchbereich!        |
        s.pos2%=INSTR(MID$(satz$,zset%,lng%(zy%+1)),srch2$)            !
      ENDIF                                                            !
    ENDIF                                                              !
    IF flg_or%                  ! OR-Modus gewählt?                    |
      IF zy%=cz%-1              ! Suchbereich = letzte Zeile?          |
        s.pos3%=INSTR(satz$,srch2$) ! OR-Begriff suchen                |
      ELSE                      ! Andere Zeile als Suchbereich!        |
        s.pos3%=INSTR(MID$(satz$,zset%,lng%(zy%+1)),srch2$)            !
      ENDIF                                                            !
    ENDIF                                                              !
    IF s.pos%>0 OR (s.pos3%>0 AND (flg_or%=1)) ! Suchbegriff bzw.      |
      '                         ! 1. ODER 2. Suchbegriff gefunden?     |
      sflg%=1                   ! Flag setzen                          |
    ELSE                                                               !
      IF ((s.pos%>0 AND s.pos2%>0) AND flg_and%=1)                     !
        '                       ! 1. UND 2. Begriff gefunden?          |
        sflg%=1                 ! Flag setzen                          |
      ELSE                                                             !
        IF MOUSEK=2 OR ck%=esc% ! Re. Maustaste o. Esc gedrückt?       |
          sflg%=1               ! Flag setzen                          |
        ENDIF                                                          !
      ENDIF                                                            !
    ENDIF                                                              !
    IF sflg%                    ! Begriff(e) gefunden oder             |
      '                         ! Suche abgebrochen?                   |
      CLR sflg%                 ! Flag löschen                         |
      undo:                     ! Label für Rücksprung                 |
      @karte_zeigen             ! Dann Karte anzeigen                  |
    ELSE                        ! Weiter!                              |
      IF mk%<2 AND ck%<>27      ! Linke oder keine Maustaste bzw.      |
        '                       ! <Esc>-Taste nicht gedrückt?          |
        GOTO nochmal            ! Dann nächste Karte (s.o.). >----'
      ENDIF
    ENDIF
```

```
  ENDIF
ELSE                              ! Blättern oder Register gewählt?
  @karte_zeigen                   ! Aktuelle Karte zeigen
ENDIF
'           Ab jetzt ist die Karte angezeigt !
'          ----------------------------------
MOUSE xp%,yp%,mk%                 ! Maus-Status holen
IF modus%=1                       ! Blättern gewählt?
  IF (mk%=1 AND (yp%<yo% OR yp%>yu%)) ! Linke Maustaste außerhalb
    '                             ! der Karte gedrückt?
    SUB card%,2                   ! Dann Kartenzähler vermindern
  ENDIF
  IF ck%=10 OR (ck%=75 AND LEN(ckey$)=2) !<Control><Return>
    '                             ! Oder Pfeil-links gedrückt?
    SUB card%,2                   ! Dann ebenfalls vermindern
  ENDIF
  IF ck%=115 AND LEN(ckey$)=2     ! <Control><Pfeil-links> gedrückt?
    SUB card%,6                   ! Kartenzähler um 5 vermindern
  ENDIF
  IF ck%=116 AND LEN(ckey$)=2     ! <Control><Pfeil-rechts> gedrückt?
    ADD card%,4                   ! Kartenzähler um 5 erhöhen
  ENDIF
ENDIF
VOID FRE(0)                       ! Garbage Collection
IF ck%=103                        ! <G> gedrückt?
  @karte_springen                 ! Dann zur Zielkarteneingabe
  IF ziel%>0                      ! Gültige Kartennummer?
    card%=ziel%                   ! Kartenzähler setzen
    modus%=1                      ! Generell Blättern einschalten
  ELSE
    IF modus%<>1                  ! Im Such- oder Registermodus?
      GOTO undo                   ! Dann gleiche Karte nochmal
    ENDIF
  ENDIF
  DEC card%                       ! Kartenzähler rücksetzen
ENDIF
IF ck%=117                        ! <U> gedrückt?
  @bildaufbau                     ! Desktop neu zeichnen
  @automatic                      ! Dann Update-Flag umschalten
  GOTO undo                       ! Und gleiche Karte nochmal
ENDIF
IF (mk% AND (yp%>yo% AND yp%<yu%)) OR ...
                        ....(sh%>0 AND (sh% AND 4)=0) OR ck%=32
  '                               ! Klick innerhalb der Karte, eine der
  '                               ! Switch-Tasten (außer <Control>)
  '                               ! Oder die <Space>-Taste gedrückt?
  @karte_aendern                  ! Dann zur Eingabe
  back3%=1                        ! Abbruch simulieren
  GOTO jump1                      ! Rücksprung
ENDIF
IF ck%=100                        ! <D> gedrückt?
  @karte_drucken                  ! Dann zur Druckerausgabe
  back3%=1                        ! Abbruch simulieren
  GOTO jump1                      ! Rücksprung
ENDIF
IF ck%=116 AND LEN(ckey$)=1       ! <T> gedrückt?
  @karte_tauschen                 ! Dann tauschen
  GOTO jump1                      ! Ggfs. Rücksprung
ENDIF
```

```
    IF ck%=105                      ! <I> gedrückt?
      @karte_import(byte1%,sg%)     ! Dann Daten importieren
      GOTO jump1                    ! Ggfs. Rücksprung
    ENDIF
    IF ck%=82                       ! <Insert> gedrückt?
      @karte_einfuegen              ! Dann einfügen
      GOTO jump1                    ! Ggfs. Rücksprung
    ENDIF
    IF ck%=127                      ! <Delete> gedrückt?
      @karte_loeschen               ! Dann löschen
      GOTO jump1                    ! Ggfs. Rücksprung
    ENDIF
    GOTO jump2                      ! Rücksprung-Abfrage übergehen
    jump1:                          ! Label für Rücksprungabfrage
    IF back3%=1                     ! Operation abbrechen?
      IF modus%<>1                  ! Im Such- oder Registermodus?
        GOTO undo                   ! Dann gleiche Karte nochmal
      ENDIF
    ELSE                            ! Operation ausführen!
      modus%=1                      ! Generell Blättern einschalten
    ENDIF
    DEC card%                       ! ...gleiche Karte nochmal zeigen
    jump2:                          !
    EXIT IF MOUSEK=3 OR ck%=esc%    ! Abbruch, wenn beide Maustasten
    '                               ! oder <Esc>-Taste gedrückt wurde(n)
    EXIT IF modus%<>1 AND (card%=karten% OR MOUSEK=2)! Abbruch, wenn
    '                               ! im Such- oder Registermodus die
    '                               ! letzte Karte erreicht ist oder die
    '                               ! rechte Maustaste gedrückt wird
    IF card%<0                      ! Karte kleiner 0?
      card%=karten%-1               ! Dann Zähler auf letzte Karte
    ENDIF
    IF card%=>karten%               ! Hinter letzter Karte?
      CLR card%                     ! Kartenzähler wieder auf Null
    ENDIF
    CLR ck%,sh%                     ! Tastenmerker klar
  LOOP
  CLR ck%,sh%                       ! Tastenmerker klar (nach EXIT IF-
Sprung)
  o.ut:
  DEFMOUSE 0
RETURN
*****************************************************************
PROCEDURE desk_taste                ! Tastaturabfrage ohne Kartenanzeige
  DO                                ! Abfrageschleife
    dkey$=INKEY$                    ! Tastatur abfragen
    dkey%=ASC(RIGHT$(dkey$))        ! ASCII der Taste ermitteln
    sh%=BIOS(11,-1)                 ! Switch-Tastenstatus holen
    IF LEN(dkey$)=1                 ! Normaltaste gedrückt?
      IF dkey%=>48 AND dkey%<=57    ! Taste von 0 bis 9 gedrückt?
        keyflg%=1                   ! Flag für Registertaste setzen
      ENDIF
      IF (sh% AND 1) OR (sh% AND 2) ! <Shift>-Taste gedrückt?
        IF dkey%=>65 AND dkey%<=90  ! Taste von A bis Z gedrückt?
          keyflg%=1                 ! Flag für Registertaste setzen
        ENDIF
      ENDIF
    ENDIF
    EXIT IF keyflg%                   ! <Shift><Registertaste> gedrückt?
```

```
    EXIT IF dkey%=98                     ! <b> oder <Help> gedrückt?
    EXIT IF dkey%=117 AND LEN(dkey$)=1 ! <u> gedrückt?
    EXIT IF dkey%=97 AND LEN(dkey$)=1  ! <a> gedrückt?
    EXIT IF dkey%=115 AND LEN(dkey$)=1 ! <s> gedrückt?
    EXIT IF dkey%=105 AND LEN(dkey$)=1 ! <i> gedrückt?
    EXIT IF dkey%=103 AND LEN(dkey$)=1 ! <g> gedrückt?
    EXIT IF dkey%=72 AND LEN(dkey$)=2  ! Aufwärts-Pfeil gedrückt?
    EXIT IF dkey%=80 AND LEN(dkey$)=2  ! Abwärts-Pfeil gedrückt?
    EXIT IF dkey%=13 OR dkey%=ASC("q") ! <Return> oder <Esc> gedrückt?
    IF dkey%=99                      ! <c> gedrückt?
      @dcolor                        ! Dann zur Farbeinstellung
    ENDIF
    EXIT IF MOUSEK=1 OR MOUSEK=2     ! Li. oder re. Maustaste gedrückt?
  LOOP
  LPOKE XBIOS(14,1)+6,0              ! Tastaturpuffer löschen
RETURN
******************************************************************
PROCEDURE card_taste                 ! Tastaturabfrage während
  '                                  ! der Kartenanzeige
  DO                                 ! Schleife.........
    ckey$=INKEY$                     ! Tastatur abfragen
    ck%=ASC(RIGHT$(ckey$))           ! ASCII der Taste ermitteln
    sh%=BIOS(11,-1)                  ! Switch-Tastenstatus holen
    EXIT IF ck%=esc%                 ! <Esc> gedrückt?
    EXIT IF ck%=32                   ! <Space> gedrückt?
    EXIT IF ck%=103 AND LEN(ckey$)=1 ! <g> gedrückt?
    EXIT IF ck%=105 AND LEN(ckey$)=1 ! <i> gedrückt?
    EXIT IF ck%=117 AND LEN(ckey$)=1 ! <i> gedrückt?
    EXIT IF ck%=100 AND LEN(ckey$)=1 ! <d> gedrückt?
    EXIT IF ck%=127 AND LEN(ckey$)=1 ! <Delete> gedrückt?
    EXIT IF ck%=82 AND LEN(ckey$)=2  ! <Insert> gedrückt?
    EXIT IF ck%=75 AND LEN(ckey$)=2  !  <- (Pfeil-links) gedrückt?
    EXIT IF ck%=77 AND LEN(ckey$)=2  !  -> (Pfeil-rechts) gedrückt?
    EXIT IF ck%=115 AND LEN(ckey$)=2 ! <Control><- gedrückt?
    EXIT IF ck%=116                  ! <Control>-> oder <T> gedrückt?
    EXIT IF sh%>0 AND (sh% AND 4)=0  ! Eine <Shift>-Taste gedrückt?
    EXIT IF MOUSEK OR ck%=13 OR ck%=10 ! Maustaste, <Return> oder
    '                                ! <Control><Return> gedrückt?
    IF ck%=72 AND LEN(ckey$)=2       ! Pfeil-hoch-Taste gedrückt?
      DPOKE 9954,MAX(yo%,MIN(yu%,(MOUSEY DIV (16/t%))*16/t%-8/t%))
      '                              ! Maus eine Zeile hoch
      OUT 4,8                        ! und neue Position aktivieren
    ENDIF
    IF ck%=80 AND LEN(ckey$)=2       ! Pfeil-runter-Taste gedrückt?
      DPOKE 9954,MAX(yo%,MIN(yu%,(MOUSEY DIV (16/t%))*16/t%+24/t%))
      '                              ! Maus eine Zeile runter
      OUT 4,8                        ! und neue Position aktivieren
    ENDIF
  LOOP
  LPOKE XBIOS(14,1)+6,0              ! Tastaturpuffer löschen
RETURN
```




```
***********************************************************************
PROCEDURE karte_zeigen                  ! Karteikartenanzeige
  DEFFILL ,0,0                          ! weiß für Hintergrund
  DEFMOUSE 0                            ! Pfeil einschalten
  shflg%=1                              ! Desktop-Restore-Flag setzen
  PBOX 8,yo%,627,yu%+24/t%+(t%-1)       !-----------.
  BOX 13,yo%+5/t%,622,yu%-5/t%                      ! Karte zeichnen
  LINE 8,yu%,627,yu%                    !-----------'
  OPEN "",#1,"VID:"                     ! VIDEO-Port auf
  rep$=CHR$(0)+CHR$(8)+CHR$(9)+CHR$(10)+CHR$(11)+CHR$(12)+CHR$(13)
  '                                     ! Steuerzeichenliste
  FOR j%=1 TO cz%                       ! Alle Eintragszeilen >------------.
    buf$=ztxt$(j%)                      ! Eintragstext puffern             !
    @xreplc                             ! Steuerzeichen eliminieren        !
    ztxt$(j%)=buf$                      ! Puffer zurück                    !
    PRINT AT(4,j%+1+yo%/16*t%);pre$(j%);": ";  ! Cursor positionieren     !
    PRINT #1,ztxt$(j%);                        ! Zeile in VID: ausgeben   !
    '                                   ! ASCIIs < 32 werden sichtbar      !
  NEXT j%                               ! Nächste Zeile    <---------------'
  DEFTEXT ,0,,6-2*(t%-1)
  IF modus%=1                           ! Blättern gewählt?
    IF d.flg%=0                         ! Allererste Kartenanzeige?
      d.flg%=1                          ! Merken
      mtx$="EINGABE = <Shift>/ EINFÜGEN=<Insert>/" !--.
      mtx$=mtx$+" LÖSCHEN=<Delete>/ IMPORT= <i>"      !
      TEXT 70+120*(t%-1),yu%+10/t%,mtx$               ! Tasten-Menü
      mtx$="GOTO KARTE= <g>/ TAUSCHEN= <t>  /"        ! schreiben
      mtx$=mtx$+" DRUCKEN=   <d>/ EXIT =  <Esc>"      !
      TEXT 70+120*(t%-1),yu%+20/t%+(t%-1),mtx$   !----'
      GET 8,yu%+2*(2-t%),627,yu%+22/t%,mline$    ! Als Bild sichern
    ELSE                                ! Menü-Bild ist schon verfügbar!
      PUT 8,yu%+2*(2-t%),mline$         ! Dann in Karte einsetzen
    ENDIF
  ELSE                                  ! Such- oder Registermodus?
```

```
    mtx$="WEITER = <Return> / DRUCKEN= <d>"   !----.
    TEXT 340+60*(t%-1),yu%+10/t%,mtx$          ! Tasten-Menü
    mtx$="EINGABE= <Shift>  / EXIT   = <Esc>"  ! schreiben
    TEXT 340+60*(t%-1),yu%+20/t%+(t%-1),mtx$ ! ----'
    IF modus%=0 AND (s.pos%>0 OR s.pos3%>0) AND zy%<cz%-1
      PRINT esc$;"p";                   ! TOS-Schrift invers
      IF s.pos%
        PRINT AT(15+s.pos%,zy%+2+yo%/16*t%); ! Cursor positionieren
        PRINT #1,srch$;                 ! 1. Suchbegriff in VID: ausgeben
      ENDIF
      IF s.pos2%
        PRINT AT(15+s.pos2%,zy%+2+yo%/16*t%); ! Cursor positionieren
        PRINT #1,srch2$;                ! 2. AND-Begriff in VID: ausgeben
      ENDIF
      IF s.pos3%
        PRINT AT(15+s.pos3%,zy%+2+yo%/16*t%); ! Cursor positionieren
        PRINT #1,srch2$;                ! 2. OR-Begriff in VID: ausgeben
      ENDIF
      PRINT esc$;"q";                   ! TOS-Schrift normal
    ENDIF
    CLR s.pos%,s.pos2%,s.pos3%          ! Suchpositionen löschen
  ENDIF
  CLOSE #1                              ! VIDEO-Port schließen
  DEFTEXT ,0,,13/t%
  TEXT 20,yu%+16/t%+(t%-1),STR$(card%) ! Karten-Nummer anzeigen
  @card_taste                           ! Tastaturabfrage
  DEFMOUSE 2                            ! Biene wieder einschalten
RETURN
**************************************************************************
PROCEDURE karte_aendern                 ! Karteneinträge ändern
  HIDEM                                 ! Maus aus (die stört hier nur)
  PAUSE 10                              ! Kleine Pause
  DPOKE VDIBASE+34,0                    ! PBOX-Rahmen aus
  PBOX 60,yu%+2*(2-t%)+1,626,yu%+20/t%+(t%-1) ! Weißen Untergrund
  DPOKE VDIBASE+34,1                    ! PBOX-Rahmen wieder an
  mtx$="UP = <Control><Return> / EXIT = <Shift><Return>"
  TEXT 244,yu%+16/t%+(t%-1),mtx$       ! Mini-Tasten-Menü schreiben
  IF ck%=32                             ! <Space>-Taste gedrückt?
    mk%=1                               ! Dann Mausklick links simulieren
  ENDIF
  IF sh% AND 8                          ! <Alternate> gedrückt?
    mk%=2                               ! Dann Mausklick rechts simulieren
  ENDIF
  OPEN "",#1,"VID:"                     ! VIDEO-Port öffnen
  DO                                    ! Eintragsdurchlauf
    CLR p.os%                           ! Adreß-Offset klar
    FOR cnt%=1 TO cz%                   ! Alle Eintragszeilen...
      IF (mk% AND (MOUSEY>yo% AND MOUSEY<yu%))
        cnt%=MAX(1,MIN(INT((MOUSEY-yo%-7)/16*t%)+1,cz%)) ! Zeilenzähler
        '                               ! aus Mausposition errechnen
        @pcalc(1)                       ! Eintrags-Offset berechnen
        IF mk%=2                        ! Rechte Maustaste gedrückt?
          ztxt$(cnt%)=STRING$(lng%(cnt%),46) ! Eintragszeile löschen
        ENDIF
        CLR mk%                         ! Maustastenmerker klar
      ENDIF
      @path(1,ztxt$(cnt%),CHR$(46)+CHR$(0),*buf$,*du$,*du$)
      PRINT AT(16,cnt%+1+yo%/16*t%);SPACE$(lng%(cnt%));" ";CHR$(174)
      PRINT AT(16,cnt%+1+yo%/16*t%);! Cursor positionieren
```

```
      FORM INPUT lng%(cnt%) AS buf$ ! Neuen Zeilentext eingeben
      sh%=BIOS(11,-1)               ! Switch-Tastenstatus holen
      IF (sh% AND 1) OR (sh% AND 2) ! Gleichzeitig mit <Return>
        '                           ! eine <Shift>-Taste gedrückt?
        o.ff%=1                     ! Eingabe-Abbruch-Flag (normal)
      ENDIF
      IF (sh% AND 8)                ! Gleichzeitig mit <Return> die
        '                           ! <Alternate>-Taste gedrückt?
        al$="      CANCEL ??|Alte Karte restaurieren?"
        ALERT 2,al$,1,"OKAY|NEIN",back2%
        IF back2%=1                 ! Letzte Änderungen ungültig?
          o.ff%=2                   ! Eingabe-Abbruch-Flag (restore)
        ELSE
          o.ff%=1                   ! Eingabe-Abbruch-Flag (normal)
        ENDIF
      ENDIF
      IF LEFT$(buf$)=esc$ OR LEFT$(buf$)=" " ! linkes Zeichen der
        '                           ! aktuellen Zeile = Esc oder Space?
        o.ff%=1                     ! Eingabe-Abbruch-Flag (normal)
        ztxt$(cnt%)=RIGHT$(buf$,LEN(buf$)-1) ! Linkes Zeichen abtrennen
        ztxt$(cnt%)=ztxt$(cnt%)+STRING$((lng%(cnt%)-LEN(buf$))+1,46)
        '                           ! Zeilenrest mit Punkten füllen
      ELSE
        IF RIGHT$(buf$)=esc$ OR RIGHT$(buf$)=" " ! Rechtes Zeichen der
          '                         ! aktuellen Zeile = Esc oder Space?
          o.ff%=1                   ! Eingabe-Abbruch-Flag (normal)
          ztxt$(cnt%)=LEFT$(buf$,LEN(buf$)-1) ! Rechtes Zeichen abtrennen
          ztxt$(cnt%)=ztxt$(cnt%)+STRING$((lng%(cnt%)-LEN(buf$))+1,46)
          '                         ! Zeilenrest mit Punkten füllen
        ELSE
          ztxt$(cnt%)=buf$+STRING$(lng%(cnt%)-LEN(buf$),46)
          '                         ! Zeile in Pufferfeld einbauen
        ENDIF
      ENDIF
      PRINT AT(16,cnt%+1+yo%/16*t%);! Cursor positionieren
      PRINT #1,ztxt$(cnt%);"  ";    ! Zeile formatiert auf VID: ausgeben
      VOID FRE(0)                   ! Garbage Collection
      BMOVE VARPTR(ztxt$(cnt%)),byte1%+p.os%,MAX(1,lng%(cnt%))
      '                             ! In Kartenspeicher übertragen
      ADD p.os%,lng%(cnt%)          ! Adreß-Offset d. nächsten Zeile
      IF sh%=4 OR sh%=20            ! Gleichzeitig mit <Return>
        '                           ! die <Control>-Taste gedrückt?
        SUB cnt%,2                  ! Zeilenzähler minus 2
        IF cnt%<0                   ! Kleiner Null?
          cnt%=MAX(1,cz%-1)         ! Zähler auf letzte Zeile
        ENDIF
        CLR p.os%                   ! Zeilen-Offset löschen
        @pcalc(0)                   ! Neues Zeilen-Offset holen
      ENDIF
      EXIT IF o.ff%>0               ! Ausstieg, wenn Eingabeabbruch
      LPOKE XBIOS(14,1)+6,0         ! Tastaturpuffer löschen
    NEXT cnt%                       ! Nächste Zeile
    EXIT IF o.ff%>0                 ! Ausstieg, wenn Eingabeabbruch
  LOOP
  CLOSE #1                          ! VIDEO-Port schließen
  VOID FRE(0)                       ! Garbage Collection
  IF o.ff%=2                        ! Restore-EXIT (<Return><Alternate>)?
    BMOVE VARPTR(satz$),byte1%,sg%  ! Alten Satz in Kartenspeicher zurück
  ELSE                              ! Normal-EXIT (<Shift><Return>) !
```

```
    buf$=SPACE$(sg%)                     ! Vergleichspuffer vorbereiten
    BMOVE byte1%,VARPTR(buf$),sg%        ! Neue Karte in Puffer schreiben
    IF buf$<>satz$                       ! Alte Karte ungleich neuer Karte?
      @update(byte1%-sta%+1,byte1%,sg%) ! Ggfs. Disk updaten
    ENDIF
  ENDIF
  VOID BIOS(11,0)                        ! Evtl. CapsLock ausschalten
  CLR o.ff%,sh%                          ! Flag und Shift-Merker löschen
  PAUSE 10                               ! Kleine Pause, damit die benutzte
  '                                      ! <Switch>-T. losgelassen werden kann
  SHOWM                                  ! Maus wieder an
RETURN
******************************************************************
PROCEDURE pcalc(diff%)                   ! Zeilen-Offsets b. Eingabe berechnen
  IF (cnt%-diff%)>0                      ! Unterhalb der ersten Zeile?
    FOR j%=1 TO (cnt%-diff%)             ! Bis zur aktuellen Zeile
      ADD p.os%,lng%(j%)                 ! Adreß-Offsets addieren
    NEXT j%                              ! Nächste Zeile
  ENDIF
RETURN
******************************************************************
PROCEDURE karte_suchen                   ! Suchbegriff(e) eingeben
  search:                                ! Start-Label für 1. Begriff
  @lbox                                  ! Eingabebox zeichnen
  PRINT AT(6,22);"1. Suchbegriff : ";
  srchmem$=srch$                         ! Letzten Such-String merken
  FORM INPUT 27 AS srch$                 ! Neuen Such-String holen
  srch$=LEFT$(srch$,MIN(25,LEN(srch$))) ! Max. 25 Zeichen
  IF LEFT$(srch$,2)=esc$+esc$ OR RIGHT$(srch$)=esc$ OR srch$=""
    '                                    ! 1. und 2. od. letztes Zeichen = Esc
    '                                    ! Oder Such-String leer?
    srch$=srchmem$                       ! Alten Such-String restaurieren
    such.exit%=1                         ! und Suchmodus abbrechen
  ELSE
    IF LEFT$(srch$)=esc$                 ! Nur erstes Zeichen = Esc?
      srch$=""                           ! Such-String löschen
      GOTO search                        ! Und zurück zur Neueingabe
    ENDIF
    PRINT AT(6,22);"1. Suchbegriff : ";srch$''
  ENDIF
  IF such.exit%=0                        ! Noch kein Such-Abbruch?
    CLR flg_and%,flg_or%                 ! AND/OR-Flags löschen
    ALERT 2,"Zweiter Suchbegriff?  ",3," AND | OR |NEIN",back3%
    IF back3%<>3                         ! 2. Suchbegriff vorgeben?
      IF back3%=1                        ! AND-Modus?
        flg_and%=1                       ! AND-Flag setzen
        PRINT AT(10,23);" U N D"         ! Modus anzeigen
      ENDIF
      IF back3%=2                        ! OR-Modus?
        flg_or%=1                        ! OR-Flag setzen
        PRINT AT(10,23);" O D E R"       ! Modus anzeigen
      ENDIF
      search2:                           ! Start-Label für 2. Begriff
      PRINT AT(6,24);"2. Suchbegriff : ";
      srch2mem$=srch2$                   ! Alten AND/OR-String merken
      FORM INPUT 27 AS srch2$            ! Neuen AND/OR-String holen
      srch2$=LEFT$(srch2$,MIN(25,LEN(srch2$))) ! Max. 25 Zeichen
      IF LEFT$(srch2$,2)=esc$+esc$ OR RIGHT$(srch2$)=esc$ OR srch2$=""
        '                                ! 1. und 2. od. letzt. Zeichen = Esc?
```

```
        srch2$=srch2mem$              ! Alten Such-String restaurieren
        PRINT AT(10,23);SPACE$(10)    ! Modusanzeige löschen
        PRINT AT(6,24);SPACE$(45)     ! Ungültige Eingabe löschen
        CLR flg_and%,flg_or%          ! AND/OR-Suche abbrechen
      ELSE
        IF LEFT$(srch2$)=esc$         ! Nur erstes Zeichen = Esc?
          srch2$=""                   ! AND/OR-String löschen
          PRINT AT(6,24);SPACE$(45)
          GOTO search2                ! Und zurück zur Neueingabe
        ENDIF
        PRINT AT(6,24);"2. Suchbegriff : ";srch2$''
      ENDIF
    ENDIF
  ENDIF
  shflg%=1                            ! Desktop-Restore-Flag setzen
RETURN
********************************************************************
PROCEDURE karte_springen              ! Sprung zu beliebiger Karte
  REPEAT                              ! Schleife...  >------.
    @lbox                             ! Eingabebox zeichnen |
    PRINT AT(14,23);"GOTO Karte Nr. : ";                    |
    FORM INPUT 5,ziel$                ! Zielposition holen  |
    ziel%=VAL(ziel$)                  ! String konvertieren |
  UNTIL ziel%<karten%+1               ! ..bis gültige Nr. <-'
  shflg%=1                            ! Desktop-Restore-Flag setzen
  @bildaufbau                         ! Menü/Register etc. zeichnen
RETURN
********************************************************************
PROCEDURE karte_tauschen
  IF hlp%=1                           ! 1. Karte schon bestimmt?
    al$="Karte Nr. "+STR$(pk%)+"|mit Karte Nr. "
    al$=al$+STR$(card%)+"|vertauschen?"
    ALERT 2,al$,1,"NEIN|OKAY",back3%
    IF back3%=2                       ! 1. mit 2. Karte tauschen?
      CLR hlp%                        ! Flag für 1. Karte löschen
      buf$=SPACE$(sg%)                ! Tauschpuffer vorbereiten
      BMOVE sta%+(pk%-1)*sg%,VARPTR(buf$),sg%       ! 1. Karte in Puffer
      BMOVE sta%+(card%-1)*sg%,sta%+(pk%-1)*sg%,sg% ! 2. in 1. Karte
      BMOVE VARPTR(buf$),sta%+(card%-1)*sg%,sg%     ! Puffer in 2. Karte
      pcrd%=(MIN(pk%,card%)-1)*s.ges%               ! Kleineres K.-Offset
      pcrd2%=((MAX(pk%,card%)-MIN(pk%,card%))+1)*sg%! Offset-Differenz
      @update(pcrd%+1,sta%+pcrd%,pcrd2%)            ! Ggfs. Disk updaten
    ENDIF
  ELSE                                ! Noch keine Tauschkarte bestimmt!
    al$="Karte Nr. "+STR$(card%)
    al$=al$+" puffern?|(nächste <T>-Taste = 2.Karte)"
    ALERT 2,al$,1,"NEIN|OKAY",back3%
    IF back3%=2                       ! 1. Tauschkarte merken?
      pk%=card%                       ! Merker = Kartennummer
      hlp%=1                          ! Flag für 1. Karte setzen
    ENDIF
  ENDIF
RETURN
********************************************************************
PROCEDURE karte_einfuegen
  LOCAL voll%                         ! 'Voll%' löschen
  al$="Leerkarte an|Position "+STR$(card%)+"|einfügen ??"
  ALERT 2,al$,1,"NEIN|OKAY",back3%
  IF back3%=2                         ! Leerkarte einfügen?
```

```
    FOR i%=sta%+kges% DOWNTO sta%+kges%-sg%+1 ! Alle Bytes der >--.
      '                                 ! hintersten Karteikarte...  |
      IF PEEK(i%)>32 AND PEEK(i%)<>46 AND PEEK(i%)<>95 ! ...evtl.    |
        '                               ! Textzeichen enthalten?     |
        voll%=1                         ! Flag setzen (Karte belegt) |
      ENDIF                                                          |
    NEXT i%                             ! Nächstes Byte <------------'
    IF voll%=1                          ! Hinterste Karte belegt?
      al$="Letzte Karteikarte|ist nicht frei !"
      ALERT 1,al$,1,"Return",back3% ! Hinweis ausgeben
    ELSE                                ! Hinterste Karte frei!
      BMOVE byte1%,byte1%+sg%,MAX(1,sg%*(karten%-card%)) ! Kartei-Rest
      '                                 ! ab aktueller Karte um eine Karte
      '                                 ! nach oben versetzen
      buf$=STRING$(sg%,46)              ! Leerkarte vorbereiten (Punkte)
      BMOVE VARPTR(buf$),byte1%,sg% ! Punktepuffer in Karte schreiben
      @update(byte1%-sta%+1,byte1%,sg%*((karten%-card%)+1)) ! Ggfs.
    ENDIF                                                  ! Disk-Update
  ENDIF
RETURN
**********************************************************************
PROCEDURE karte_loeschen
  ALERT 2,"Karte "+STR$(card%)+" löschen ??",1,"NEIN|OKAY",back3%
  IF back3%=2                           ! Aktuelle Karte löschen?
    BMOVE byte1%+sg%,byte1%,MAX(1,sg%*(karten%-card%)) ! Kartei-Rest
    '                                   ! ab nächster Karte um eine Karte
    '                                   ! nach unten versetzen
    buf$=STRING$(sg%,46)                ! Leerkarte vorbereiten (Punkte)
    BMOVE VARPTR(buf$),sta%+kges%-sg%,sg% ! Punktepuffer in hinterste
    '                                   ! Karte schreiben
    @update(byte1%-sta%+1,byte1%,sg%*((karten%-card%)+1)) ! Ggfs.
  ENDIF                                                  ! Disk-Update
RETURN
**********************************************************************
PROCEDURE karte_drucken                 ! Druckerausgabe d. aktuellen Karte
  IF OUT?(0)=TRUE                       ! Drucker an, bzw. OnLine?
    LPRINT                              ! Leerzeile
    FOR k%=1 TO cz%                     ! .---> Alle Eintragszeilen >--.
      LPRINT pre$(k%);": ";ztxt$(k%)! |     ...schreiben...          |
    NEXT k%                             ! '----< Nächste Zeile <-------'
  ELSE                                  ! Drucker antwortet nicht!
    ALERT 1,"Drucker nicht empfangsbereit!",1,"Weiter",back2%
  ENDIF
RETURN
**********************************************************************
PROCEDURE karte_import(impad%,impln%)  ! Externe Daten importieren
  IF impln%=kges%                       ! Importlänge = Karteilänge?
    al$="Kartei ab Start mit|Import-Datei überschreiben?"
    ALERT 2,al$,1,"NEIN|OKAY",back3%
  ELSE                                  ! Importlänge kleiner Kartei !
    al$="Nur aktuelle Karte oder|ab aktueller Karte mit|"
    al$=al$+"Import-Datei überschreiben?"
    ALERT 2,al$,1,"NEIN| Karte |ab Karte",back3%
    IF back3%=3                         ! Ab aktueller Karte bis Karteiende
      impln%=kges%-(byte1%-sta%)        ! Verbleibende Karteilänge
    ENDIF
  ENDIF
  IF back3%>1                           ! Daten tatsächlich importieren?
    @head(1,"Import-Datei wählen")
```

```
    FILESELECT "\*.*","",import$     ! Dateiauswahl
    @head(0,"")
    IF EXIST(import$)                ! Gewählte Datei existiert?
      OPEN "I",#1,import$            ! Dann öffnen
      BGET #1,impad%,MIN(impln%,LOF(#1)) ! Datei ab gewählter Adresse
      '                              ! Bis max. Karteiende laden
      CLOSE #1                       ! File schließen
      @update(impad%-sta%+1,impad%,impln%) ! Ggfs. Disk-Update
    ENDIF
  ENDIF
RETURN
*******************************************************************
PROCEDURE sorter                     ! Kartei sortieren
  al$=" Kartei nach|Eintragszeile "+STR$(zy%+1)+"|>> "
  al$=al$+pre$(zy%+1)+" <<|  sortieren?"
  ALERT 2,al$,1,"NEIN|OKAY",back3%
  IF back3%=2                        ! Sortieren?
    DEFMOUSE 2                       ! Bienen-Maus an
    vh=22/karten%                    ! Verhältnis der Menüzeilenhöhe zur
    '                                ! Karteilänge (Busy-Effekt s.u.)
    buf$=SPACE$(sg%)                 ! Kartenpuffer vorbereiten
    FOR j%=0 TO karten%-1            ! Alle Karten
      sbuf1$=STRING$(lng%(zy%+1),0)  ! 1. String-Puffer nullen
      sbuf2$=sbuf1$                  ! 2. Puffer = 1.Puffer
      BMOVE sta%+(j%*sg%)+(zset%-1),VARPTR(sbuf1$),MAX(1,lng%(zy%+1))
      '                              ! 1. Karte(nzeile) in 1. Puffer
      LINE 100,(116+j%*vh)/t%,110,(116+j%*vh)/t% ! Busy-Effekt
      FOR k%=j%+1 TO karten%-1       ! Alle (Vergleichs-)Karten
        BMOVE sta%+(k%*sg%)+(zset%-1),VARPTR(sbuf2$),MAX(1,lng%(zy%+1))
        '                            ! 2. Karte(nzeile) in 2. Puffer
        IF sbuf1$=STRING$(lng%(zy%+1),46) ! 1. Karte(nzeile) nur Punkte?
          sbuf1$=STRING$(lng%(zy%+1),255) ! Dann ganz nach oben setzen
        ENDIF
        IF sbuf2$=STRING$(lng%(zy%+1),46) ! 2.Karte(nzeile) nur Punkte?
          sbuf2$=STRING$(lng%(zy%+1),255) ! Dann ganz nach oben setzen
        ENDIF
        IF sbuf1$<>sbuf2$            ! 1. String ungleich 2. String?
          IF UPPER$(sbuf1$)>UPPER$(sbuf2$) ! 1. String > 2. String?
            SWAP sbuf1$,sbuf2$       ! Puffer austauschen
            BMOVE sta%+(j%*sg%),VARPTR(buf$),sg%  !-----. Speicher-
            BMOVE sta%+(k%*sg%),sta%+(j%*sg%),sg% !-----| Blöcke
            BMOVE VARPTR(buf$),sta%+(k%*sg%),sg%  !-----' austauschen
          ENDIF
        ENDIF
      NEXT k%                        ! Nächste Vergleichskarte(nzeile)
    NEXT j%                          ! Nächste Sortierkarte(nzeile)
    @update(1,sta%,kges%)            ! Ggfs. Disk-Update
    DEFMOUSE 0                       ! Pfeil-Maus wieder an
  ENDIF
RETURN
*******************************************************************
PROCEDURE liste
  buf$=SPACE$(MAX(1,lng%(zy%+1)))    ! Zeilenpuffer vorbereiten
  REPEAT                             ! Schleife...  >------.
    @lbox                            ! Eingabebox zeichnen |
    PRINT esc$;"p";                  ! Schrift invers      |
    PRINT AT(6,23);">> ";pre$(zy%+1);" <<";esc$;"q";"  Liste ab Karte Nr.
: ";
    FORM INPUT 5 AS strt$            ! Startposition holen |
```

```
    EXIT IF LEFT$(strt$)=esc$         ! Abbruch wenn Esc       !
    strt%=VAL(strt$)                  ! String konvertieren    !
    IF strt$=""                       ! Ohne Eingabe zurück    !
      strt%=1                         ! Dann ab Karte 1        !
    ENDIF                                                      !
  UNTIL strt%<karten%+1 AND strt%>0 ! ..bis gültige Nr. <-'
  IF LEFT$(strt$)<>esc$               ! 1. Zeich. d. Nummerneingabe <>'Esc'
    OPEN "",#1,"VID:"                 ! VIDEO-Port auf
    CLS                               ! Bildschirm klar
    PRINT esc$;"p";                   ! Schrift invers
    PRINT "Liste: '";pre$(zy%+1);"'";SPACE$(18);
    PRINT "(<Taste> = nächste Zeile / <Esc> = Abbruch)";
    PRINT STRING$(80,"=");esc$;"q"; ! Unterstreichen und Schrift normal
    rep$=CHR$(0)+CHR$(8)+CHR$(9)+CHR$(10)+CHR$(11)+CHR$(12)+CHR$(13)
    '                                 ! Steuerzeichenliste
    FOR j%=strt%-1 TO karten%-1       ! Alle Karten ab Startnummer >----.
      BMOVE sta%+(j%*sg%)+(zset%-1),VARPTR(buf$),MAX(1,lng%(zy%+1))    !
      '                               ! In Puffer übertragen            !
      @xreplc                         ! Steuerzeichen eliminieren       !
      PRINT esc$;"p";j%+1;SPACE$(6-LEN(STR$(j%+1)));"¦";esc$;"q";" "; !
      PRINT #1,buf$;                  ! Zeile in VID: ausgeben          !
      PRINT                           ! CR/LF                           !
      IF INP(2)=27                    ! <Esc> gedrückt?                 !
        j%=karten%-1                  ! Dann Zähler auf Endwert         !
      ENDIF                                                             !
      LPOKE XBIOS(14,1)+6,0           ! Tastaturpuffer löschen          !
    NEXT j%                           ! Nächste Karte  <---------------'
    PRINT esc$;"p";                   ! Schrift invers
    PRINT SPACE$(MAX(1,lng%(zy%+1)));"<Return>"
    PRINT esc$;"q";                   ! Schrift normal
    REPEAT                            ! auf...
    UNTIL INP(2)=13                   ! ... <Return> warten
    CLOSE #1                          ! VIDEO-Port schließen
  ELSE                                ! Nummerneingabe m. Esc abgebrochen!
    strt$=RIGHT$(strt$,LEN(strt$)-1)! Eingabe-String korrigieren ENDIF
  CLS                                 ! Bildschirm klar
  shflg%=1                            ! Flag für Desktop-Hintergrund
  @bildaufbau                         ! Desktop restaurieren
RETURN
*********************************************************************
PROCEDURE automatic                   ! Automatische Update-Kontrolle
  auto%=auto% XOR 1                   ! UPDATE-An/Aus-Flag umschalten
  IF auto%=1                          ! Flag auf "aus"?
    al$="Automatische Daten-Sicherung|wird ausgeschaltet !"
    ALERT 1,al$,1,"OKAY",back3%
  ELSE                                ! Flag auf "an"
    al$="Änderungen werden|automatisch auf|Disk gesichert !"
    ALERT 1,al$,1,"OKAY",back3%
  ENDIF
  DEFFILL 1,0,0
  PBOX 16,144/t%-1,161,165/t%-1       ! UPDATE-Menüpunkt löschen
  @stext(20,159/t%,0,"UPDATING "+CHR$(32+10*(1-auto%))+"  <u>",2,2)
  GET 10,56/t%,170,254/t%,m.bild$     ! Gesamtes Menübild erneuern
  CLR wahl%,menue%                    ! Menüauswahl rücksetzen
  @service                            ! Ggfs. Disk-Update
RETURN
*********************************************************************
PROCEDURE service
  IF change%=1 AND auto%=0            ! Karte(n) geändert und UPDATE "an"?
```

```
    al$="Alle bisherige(n)¦Änderung(en) auf¦Diskette sichern?"
    ALERT 2,al$,1,"OKAY¦NEIN",back2%
    IF back2%=1                         ! Bisherige Änderungen sichern?
      @tiptext("Aktuelle Kartei wird gesichert !") ! Kleiner Hinweis
      CLR change%                       ! Änderungen-Flag aus
      @update(1,sta%,kges%)             ! Änderungen auf Disk sichern
      shflg%=1                          ! Flag für Desktop-Hintergrund
      @bildaufbau                       ! Desktop zeichnen
    ENDIF
  ENDIF
RETURN
*******************************************************************
PROCEDURE update(fseek%,crdad%,blkln%) ! Änderungen auf Disk sichern
  IF auto%=0                            ! UPDATING "an"?
    REPEAT                              ! Schleife...     >-----------.
      CLR back2%                        ! Exit-Flag löschen           !
      IF EXIST(datei$)                  ! Disk nicht gewechselt?      !
        IF fseek%=1 AND crdad%=sta% AND blkln%=kges%                  !
          '                             ! Ganze Kartei komplett sichern?
          BSAVE datei$,sta%,kges%       ! Dann sichern                !
        ELSE                            ! Teilkartei sichern!         !
          OPEN "U",#1,datei$            ! Kartei-File öffnen          !
          SEEK #1,fseek%-1              ! File-Pointer auf Karten-Offset
          BPUT #1,crdad%,blkln%         ! geänderte(n) Karte(n) sichern
          CLOSE #1                      ! Datei schließen             !
        ENDIF                                                         !
      ELSE                              ! Disk wurde gewechselt !     !
        al$=datei$+"¦nicht gefunden !¦Datendisk einlegen !"           !
        ALERT 1,al$,1,"OKAY",back2%                                   !
      ENDIF                                                           !
    UNTIL back2%=0                      ! ...bis ohne Fehler fertig <----'
  ELSE
    change%=1
  ENDIF
RETURN
*******************************************************************
PROCEDURE lbox
  DEFFILL ,2,8                        !----------.
  PBOX 32,332/t%,416,390/t%                      !  Eingabebox für
  DEFFILL ,0,0                                   !- GOTO, User-Code
  PBOX 30,330/t%,414,388/t%                      !  und Sucheingabe
  BOX 32,332/t%,412,386/t%            !----------'
RETURN
*******************************************************************
PROCEDURE bildaufbau                  ! Desktop komplett zeichnen
  IF shflg%=1                         ! Auch Hintergrund erneuern?
    @desk                             ! Dann zeichnen
    CLR shflg%                        ! Flag wieder löschen
  ENDIF
  PUT 10,11/t%,tt$                    ! Kartei-Titel-Image zeichnen
  PUT 10,56/t%,m.bild$                ! Menü-Image zeichnen
  PUT 538,370/t%-cz%*10/t%,zeilen$    ! Eintragszeilen-Image zeichnen
  GRAPHMODE 3                         ! XOR-Modus
  DEFFILL ,2,8                        ! DEFFILL schwarz
  PBOX 541,385/t%-cz%*10/t%+zy%*10/t%-(t%-1),...
                        ...626,395/t%-cz%*10/t%+zy%*10/t%
  '                                   ! Aktuelle Eintragszeile revers
  GRAPHMODE 1                         ! REPLACE-Modus
  @register                           ! Register zeichnen
```

```
 DEFFILL ,0,0                          ! DEFFILL wieder weiß
RETURN
*******************************************************************
PROCEDURE desk                         ! Desktop-Hintergrund zeichnen
 GRAPHMODE 1
 DEFFILL ,2,8
 PBOX 7,7/t%,636,396/t%
 DEFFILL ,2,4
 PBOX 3,3/t%,632,392/t%
 DEFFILL ,0,0
RETURN
*******************************************************************
PROCEDURE register                     ! Register A - Z und 0 - 9 zeichnen
 DEFTEXT ,17,,6-2*(t%-1)               ! Textgröße an Auflösung anpassen
 FOR i%=25 DOWNTO 0                    ! 26 Buchstaben
   PUT 55+i%*20,335/t%-i%*13/t%,cd$              ! Register-Image zeichnen
   TEXT 110+i%*20,345/t%-i%*13/t%,CHR$(65+i%) ! Buchstaben schreiben
 NEXT i%                                         ! Nächster Buchstabe
 FOR i%=9 DOWNTO 0                               ! 10 Ziffern
   PUT 240+i%*20,335/t%-i%*13/t%,cd$             ! Image zeichnen
   TEXT 295+i%*20,345/t%-i%*13/t%,STR$(i%)       ! Ziffer schreiben
 NEXT i%                                         ! Nächste Ziffer
 DEFTEXT ,0,,13/t%                               ! Normal-Text
RETURN
*******************************************************************
PROCEDURE intro                        ! Programm vorbereiten
 t%=MIN(2,3-XBIOS(4))                  ! Y-Auflösungsteiler
 r_xt%=2-SGN(XBIOS(4))                 ! X-Auflösungsteiler für Hilfs-Procs
 r_yt%=MIN(2,3-XBIOS(4))               ! Y-Auflösungsteiler für Hilfs-Procs
 @rahmen                               ! Pseudo-PBOX
 DEFTEXT ,,,12/t%                      ! Textgröße anpassen
 @stext(32,24/t%,576,"VARIO-RAM-KART",6,4/t%) !- Screen-Aufbau
 DEFTEXT ,1,,6-2*(t%-1)                ! Andere Textart und -größe
 @stext(120,386/t%,400,"'Das große GFA BASIC Buch' (DATA BECKER)",3,2)
 DEFTEXT ,0,,13/t%                     ! Normal-Text
 al$="Datei bearbeiten|oder neu definieren?"
 ALERT 2,al$,1," OPEN |DEFINE",back%
 IF back%=2                            ! Neue Kartei einrichten?
   CLS                                 ! Bildschirm klar
   @define                             ! und Definiton aufrufen
 ENDIF
 esc%=27                               ! Escape-Code
 esc$=CHR$(esc%)                       !  -"-   -"-
 suchen%=1                             !-----.
 blaettern%=2                                !
 sortieren%=3                                !  Menü-Opcodes
 update%=4                                   !- definieren
 springen%=5                                 !
 import%=6                                   !
 progexit%=7                           !-----'
 zset%=1                               ! Zeilen_String-Offset auf 1
 @head(1,"Kartei wählen ('.DEF'-Extension)")
 REPEAT                                ! .DEF-Datei-Auswahl >------------.
   REPEAT                              ! Fileselect-Schleife >--------.  !
     FILESELECT "\*.DEF","________.DEF",sel$                          !  !
   UNTIL EXIST(sel$) OR sel$=""        ! Bis Auswahl gültig <---------'  !
   IF RIGHT$(sel$,4)<>".DEF"           ! Falsche Extension?              !
     IF sel$>""                        ! OKAY angeklickt?                !
       ALERT 3,"Keine '.DEF'-Datei !!",1,"Nochmal",back2%                !
```

```
    ELSE                            ! ABBRUCH angeklickt!          !
      EDIT                          ! Dann Ende                    !
    ENDIF                                                          !
  ENDIF                                                            !
UNTIL back2%=0                      ! bis .DEF-Datei gewählt wurde <--'
@head(0,"")
OPEN "I",#1,sel$                    ! .DEF-Datei öffnen
INPUT #1,datei$,titel$,karten%,cz% ! Defs einlesen
DIM lng%(cz%),pre$(cz%),ztxt$(cz%) ! Felder
FOR i%=1 TO cz%                     ! für alle Eintragszeilen >---------.
  INPUT #1,lng%(i%),pre$(i%)        ! Länge und Titel lesen             !
  ADD sg%,lng%(i%)                  ! Alle Zeichen einer Karte addieren !
  ztxt$(i%)=SPACE$(lng%(i%))        ! Zeilenpuffer                      !
NEXT i%                             ! Nächste Zeile <-------------------'
CLOSE #1                            ! .DEF-Datei schließen
kges%=sg%*karten%                   ! Gesamte Karteilänge
satz$=SPACE$(sg%)                   ! Kartenpuffer
IF FRE(0)-50000<kges%               ! Kartei größer freier Speicher?
  ALERT 3,"Speicher nicht ausreichend !",1,"Abbruch",back%
  @abbruch                          ! Ende/Neustart/Definition
ENDIF
ON BREAK GOSUB abbruch              ! GFA-Abruchfunktion abfangen
RESERVE 30000                       ! 16 KB für Fileselect-Box,
'                                   !+10 KB für Menü/Register/Titel
'                                   ! +2 KB für Suchpuffer
'                                   ! +2 KB Toleranz
sta%=HIMEM+20000                    ! +20000 für V2.02 zur Sicherheit
@rahmen                             ! Pseudo-Pbox
yo%=((INT(((24-cz%)/2)-1)*16)+7)/t% ! Karten-Oberkante  --. Auflösungs-
yu%=yo%+16/t%+cz%*16/t%             ! Karten-Unterkante --' abhängig
RETURN
******************************************************************
PROCEDURE datload                   ! Kartei laden
  lader:                            ! Start-Label für Lader
  VOID FRE(0)                       ! Garbage Collection
  IF EXIST(datei$)                  ! .DAT-Datei vorhanden?
    @tiptext("Datei wird geladen !")! Kleiner Hinweis
    OPEN "I",#1,datei$              ! Kartei-Daten-File öffnen
    BGET #1,sta%,MIN(LOF(#1),sg%)   ! 1. Karte laden (max. File-Größe)
    CLOSE #1                        ! Daten-File wieder löschen
    IF PEEK(sta%+sg%-1)=ASC(">")    ! Letztes Zeichen der ersten
      '                             ! Karte ist >?
      IF PEEK(sta%+sg%-4)=ASC("<")  ! 4. letztes Zeichen der ersten
        '                           ! Karte ist <?
        ON BREAK CONT               ! GFA-Abbruchfunktion unterdrücken
        REPEAT                      !---> UserCode-Eingabeschleife >----.
          @lbox                     ! Eingabebox zeichnen               !
          PRINT AT(10,23);"Bitte User-Code eingeben : ";                !
          FORM INPUT 5,code$        ! Zielposition holen                !
          code=VAL(code$)           ! String in value konvertieren      !
          IF LEFT$(code$)=esc$      ! Erstes String-Zeichen = Esc?      !
            RESERVE XBIOS(2)-HIMEM-16384+FRE(0) ! Speicher-Restore      !
            EDIT                    ! Programmende                      !
          ENDIF                                                         !
        UNTIL code=PEEK(sta%+sg%-2)+PEEK(sta%+sg%-3)*2^8 ! Eingegebener !
        '                           !----< Code = Karteischlüssel?  <---'
        ON BREAK GOSUB abbruch      ! Dann Abbruchfunktion wieder an
        CLS                         ! Bildschirm klar
        @rahmen                     ! Pseudo-PBOX
```

```
      @tiptext("Datei wird geladen !")
    ENDIF
  ENDIF
  OPEN "I",#1,datei$                ! Kartei-Daten-File öffnen
  BGET #1,sta%,MIN(LOF(#1),kges%) ! Komplette Kartei laden
  '                                 ! (max. Kartei- bzw. File-Länge)
  CLOSE #1
 ELSE                               ! .DAT-Datei nicht gefunden!
  al.str$="Datei nicht gefunden !|Neu einrichten oder"
  al.str$=al.str$+"|auf anderer Disk suchen?"
  ALERT 1,al.str$,2,"NEUE DAT|NEUE DSK|ABBRUCH",dummy%
  IF dummy%=1                       ! Datei neu einrichten?
    @tiptext("Moment bitte !")      ! Kleiner Hinweis
    fuell$=STRING$(500,46)          ! Füllpuffer mit Punkten
    FOR i%=sta% TO sta%+kges% STEP 500 ! Gesamten Dateipuffer
      BMOVE VARPTR(fuell$),i%,500 ! Mit Punkten füllen
    NEXT i%
    BSAVE datei$,sta%,kges%         ! Präparierte Datei speichern
  ENDIF
  IF dummy%=2                       ! Diskette gewechselt?
    GOTO lader                      ! Dann nochmal versuchen
  ENDIF
  IF dummy%=3                       ! Abbruch?
    RESERVE XBIOS(2)-HIMEM-16384+FRE(0) ! Speicher restaurieren
    EDIT                            ! Und Ende
  ENDIF
 ENDIF
 @desk                              ! Desktop-Hintergrund zeichnen
 @bild                              ! Erster Bildaufbau
 @bildaufbau                        ! Desktop komplett zeichnen
 DEFTEXT ,0,,13/t%                  ! Normal-Text
RETURN
*******************************************************************
PROCEDURE bild                      ! Einmaliger Bildaufbau
 SETCOLOR 1,0,7,0                   ! Farben für Midres setzen
 SETCOLOR 2,7,0,0                   !  -"-   -"-   -"-   -"-
 DEFFILL 3,2,8                      !--------------.
 PBOX 12,13/t%,406,50/t%                           !  Kartei-
 DEFFILL 1,2,2                                     !--Desktop
 PBOX 10,11/t%,404,48/t%                           !  zeichnen
 DEFTEXT ,4,,26/t%                                 !
 @stext(16,40/t%,385,titel$,4,4)    !--------------'
 GET 10,11/t%,406,50/t%,tt$         ! Und als Bild puffern
 GRAPHMODE 1                        ! REPLACE-Modus
 DEFFILL 1,2,8                      !         -----.
 PBOX 12,58/t%,170,254/t%                          !
 DEFFILL ,0,0                                      !
 PBOX 10,56/t%,168,252/t%                          !
 BOX 12,56/t%+2,166,252/t%-2                       !
 FOR i%=63 TO 225 STEP 27                          !
   BOX 16,i%/t%-1,162,(i%+21)/t%                   !-   Menü
   PBOX 16,i%/t%-1,161,(i%+21)/t%-1                !  zeichnen
 NEXT i%                                           !
 DEFTEXT 3,0,,13/t%                 ! Normal-Text  !
 RESTORE m.text                     ! DATA-Zeiger  !
 FOR i%=78 TO 240 STEP 27           ! Menüzeilen   !
   READ m.tx$                       ! lesen        !
   @stext(20,i%/t%,0,m.tx$,2,2)     ! schreiben    !
 NEXT i%                            !         -----'
```

```
 m.text:                           ! Menü-Data-Label
 DATA SUCHEN     <s>,BLAETTERN   <b>,SORTIEREN   <a>
 DATA UPDATING * <u>,GOTO KARTE  <g>,IMPORT      <i>
 DATA Q U I T    <q>
 GET 10,56/t%,170,254/t%,m.bild$   ! Menü als Bild puffern
 GRAPHMODE 1                       !--------------.
 DEFTEXT ,17,,6-2*(t%-1)                          !
 DEFFILL ,2,8                                     !
 PBOX 558,13/t%,629,61/t%                         !
 DEFFILL ,0,0                                     !
 PBOX 555,10/t%,626,58/t%                         !- Registerkarte
 BOX 567,21/t%,585,12/t%                          !  zeichnen
 BOX 569,19/t%,583,14/t%                          !
 PBOX 595,11/t%,621,45/t%          !--------------'
 GET 555,10/t%,629,61/t%,cd$       ! Und als Bild puffern
 DEFFILL ,2,8                      !--------------.
 PBOX 538,370/t%-cz%*10/t%,630,390/t%             !
 DEFFILL ,0,0                                     !
 PBOX 538,370/t%-cz%*10/t%,628,388/t%             !
 BOX 540,370/t%-cz%*10/t%+2/t%,626,380/t%-cz%*10/t%+2/t%
 BOX 540,380/t%-cz%*10/t%+4/t%,626,388/t%-2/t%    !  Eintragszeilen-
 DEFTEXT ,0,,6/t%+(t%-1)                          !- Anzeige zeichnen
 GRAPHMODE 2                                      !
 TEXT 544,386/t%-10/t%*cz%-6/t%,"Try <HELP>"      !
 FOR i%=cz% DOWNTO 1                              !
   TEXT 544,386/t%-10/t%*(cz%-i%)-3/t%,pre$(i%)   !
 NEXT i%                           !--------------'
 GET 538,370/t%-cz%*10/t%,630,390/t%,zeilen$ ! Und als Bild puffern
 GRAPHMODE 1
RETURN
**************************************************************************
PROCEDURE abbruch                  ! Programmende
 IF auto%<>0                       ! UPDATING "aus"?
   CLR auto%                       ! Dann wieder "an"
   @service                        ! Und ggfs. Disk-Update
   auto%=1                         ! UPDATING wieder "aus"
 ELSE                              ! UPDATING ist "an"!
   @service                        ! Dann ggfs. Disk-Update
 ENDIF
 ALERT 2,"Neue Kartei laden?",3,"OKAY|NEIN|CONT",dummy
 IF dummy<>3                       ! Kein CONT?
   RESERVE XBIOS(2)-HIMEM-16384+FRE(0) ! Speicher restaurieren
   IF dummy=1                      ! Neue Kartei?
     RUN                           ! Dann Neustart
   ENDIF
   IF dummy=2                      ! Abbruch?
     EDIT                          ! Dann Ende
   ENDIF
 ENDIF
RETURN
**************************************************************************
PROCEDURE define                   ! Kartei einrichten
 DIM bez$(20),ln$(20),ln%(20)      ! Eingabe-Felder dimensionieren
 VOID BIOS(11,16)                  ! CapsLock einschalten
 FOR cz%=1 TO 20                   ! Max. 20 Zeilen pro Karte
   PRINT "TITEL für Zeile ";cz%'"(<Control><Return>=EINGABE-ENDE) :  "'
   VOID BIOS(11,16)                ! CapsLock ggfs. wieder einschalten
   FORM INPUT 10 AS bez$(cz%)      ! Zeilentitel (max. 10 Zeichen)
   bez$(cz%)=bez$(cz%)+SPACE$(10-LEN(bez$(cz%))) ! mit Space auffüllen
```

```
    EXIT IF BIOS(11,-1)=4 OR BIOS(11,-1)=20 ! Abbruch wenn
    '                                   ! <Control><Return> oder
    '                                   ! <Control><Return> und CapsLock
    REPEAT                              ! Eingabeschleife für Länge >----.
      PRINT "Länge der Eintragszeile ";cz%'" (max. 60) :  "'        !
      FORM INPUT 2,ln$(cz%)             ! Max. 2 Zeichen eingeben       !
      ln%(cz%)=VAL(ln$(cz%))            ! String in value konvertieren  !
    UNTIL ln%(cz%)>0 AND ln%(cz%)<61 ! Bis gültige Länge <-----------'
  NEXT cz%                              ! Nächste Zeile
  DEC cz%                               ! Zeilenzähler -1
  CLS                                   ! Bildschirm löschen
  PRINT "Disk-File-Name der Datei (max. 8 Zeichen) :";
  FORM INPUT 8,datei$                   ! Kartei-File-Namen eingeben
  PRINT "Datei-Überschrift :";
  INPUT titel$                          ! Kartei-Desktop-Titel eingeben
  REPEAT                                ! Eingabe-Schleife ---.
    PRINT "Karteikarten-Anzahl dieser Datei :";                 !
    INPUT karten%                                               !
  UNTIL karten%>0                       ! Für Kartenanzahl ---'
  VOID BIOS(11,0)                       ! CapsLock wieder ausschalten
  OPEN "O",#1,datei$+".DEF"             ! .DEF-Datei öffnen
  datei$=datei$+".DAT"                  ! File-Namen+.DAT
  WRITE #1,datei$,titel$                ! -----.
  WRITE #1,karten%,cz%                         ! Alle Defs
  FOR i%=1 TO cz%                              ! in .DEF-Datei
    WRITE #1,ln%(i%),bez$(i%)                  ! schreiben
  NEXT i%                               ! -----'
  CLOSE #1                              ! Datei schließen
  CLS                                   ! Bildschirm löschen
  ERASE bez$()                          ! ---.
  ERASE ln$()                               ! Felder löschen
  ERASE ln%()                           ! ---'
RETURN
*********************************************************************
PROCEDURE blinker(b.flg%,bxl%,byo%,bxr%,byu%)
  ' Invertiert Bildschirmbereich bzw. läßt ihn einmal blinken.
  '          (auch für Blitter-TOS!)
  ' B.flg%   : Flag, ob invertiert (1) oder geblinkt (0) werden soll
  ' Bxl%, Byo%, Bxr% und Byu%:
  '            Koordinatenübergabe in Hires.
  '            Midres/Lowres werden angepaßt.
  '
  DEFFILL 1,2,8                         ! DEFFILL schwarz
  GRAPHMODE 3                           ! XOR-Modus
  PBOX bxl%/r_xt%,byo%/r_yt%,bxr%/r_xt%,byu%/r_yt%   ! Pbox invers
  IF b.flg%=0                           ! Blinker "an"?
    PAUSE 10                            ! Klitztekleine Pause
    PBOX bxl%/r_xt%,byo%/r_yt%,bxr%/r_xt%,byu%/r_yt% ! Pbox nochmal..
  ENDIF                                              ! ..invers
  GRAPHMODE 1                           ! GRAPHMODE und DEFFILL ggfs. ..
RETURN                                  ! ...hinterher restaurieren
*********************************************************************
PROCEDURE head(h.flg%,h.txt$)
  ' Zeichnet in allen Auflösungen eine graue Box mit vorgegebenem
  ' Text über einer Fileselect-Box.
  ' =======> Die Prozedur Head finden Sie unter FILESELECT.
RETURN
*********************************************************************
```

```
PROCEDURE tiptext(tip$)
 ' Zentriert beliebigen Text in Bildschirmmitte
 LOCAL tln%
 GRAPHMODE 1                          ! REPLACE-Modus
 DEFFILL ,0,0                         ! DEFFILL weiß
 DEFTEXT ,0,,13/t%                    ! Normal-Text
 tln%=LEN(tip$)*4                     ! Halbe String-Pixel-Länge
 PBOX 320/r_xt%-tln%-6,190/r_yt%-5,320/r_xt%+tln%+6,220/r_yt%+5 !--.
 BOX 320/r_xt%-tln%-3,190/r_yt%-2,320/r_xt%+tln%+3,220/r_yt%+2     ! Box
 BOX 320/r_xt%-tln%-3,190/r_yt%-2,320/r_xt%+tln%+4,220/r_yt%+3  !--'
 TEXT 320/r_xt%-tln%,212/r_yt%,tip$ !Text zentriert ausgeben
 GRAPHMODE 1                          ! GRAPHMODE, DEFFILL und DEFTEXT...
RETURN                                ! ...ggfs. hinterher restaurieren
***************************************************************************
PROCEDURE rahmen
 ' Zeichnet eine mit einem Zufallsmuster gefüllte Box in
 ' Einheitsgröße (nur Midres/Hires - auch für Blitter-TOS!)
 '
 LOCAL z$,j%,i%
 FOR j%=1 TO 16/t%                    ! 8 (Midres) oder 16 (Hires)
   must$=CHR$(RANDOM(128))+CHR$(RANDOM(128)) ! Zufall-Words
   z$=z$+MKL$(0)+STRING$(36+40*(r_yt%-1),must$)+MKL$(0) ! plus je ein
 NEXT j%                              ! Null-Longword vorn und hinten
 FOR i%=32/r_yt% TO 360/r_yt% STEP 16/r_yt%/r_yt% ! jeweils 4 (Midres)
   '                                  ! bzw. 16 (Hires) Bildschirmzeilen
   BMOVE VARPTR(z$),XBIOS(2)+i%*80*r_yt%,1280/r_yt% ! mit Muster füllen
 NEXT i%                              ! Nächster Zeilenblock
 BOX 7/r_yt%,7/r_yt%,638-7/r_yt%,393/r_yt%     !-------.
 BOX 31/r_yt%,31/r_yt%,638-31/r_yt%,369/r_yt%          ! Und Rahmen
 BOX 7/r_yt%,7/r_yt%,639-7/r_yt%,394/r_yt%             ! drumherum
 BOX 31/r_yt%,31/r_yt%,639-31/r_yt%,370/r_yt%  !-------'
RETURN
***************************************************************************
PROCEDURE stext(s.xt%,s.yt%,s.xl%,s.txt$,s.xo%,s.yo%)
 ' Gibt Schattentext mit aktuellen Textatrributen an
 ' beliebiger Bildschirmposition aus.
 '
 ' S.xt%, S.yt%, S.xl% und S.txt$:
 '          Siehe DEFTEXT-Parameter 1-4
 ' S.xo%' und S.yo%:
 '          Schatten- X/Y-Offsets in Pixel
 '
 GRAPHMODE 2                          ! TRANSPARENT-Modus
 TEXT s.xt%-1,s.yt%,s.xl%,s.txt$    !-------.
 TEXT s.xt%+1,s.yt%,s.xl%,s.txt$           !- Textumrandung zeichnen
 TEXT s.xt%,s.yt%-1,s.xl%,s.txt$           !
 TEXT s.xt%,s.yt%+1,s.xl%,s.txt$    !-------'
 TEXT s.xt%+s.xo%,s.yt%+s.yo%,s.xl%,s.txt$ ! Textschatten zeichnen
 GRAPHMODE 3                          ! XOR-Modus
 TEXT s.xt%,s.yt%,s.xl%,s.txt$       ! Text in Umrandung einsetzen
 GRAPHMODE 1                          ! GRAPHMODE...
RETURN                                ! ...ggfs. hinterher restaurieren
***************************************************************************
PROCEDURE path(p.flg%,p.str$,p.sgn$,p.adr%,d.adr%,f.adr%)
 ' Untersucht einen String ab String-Ende rückwärts auf
 ' das erste Vorkommen eines in einer Liste vorgegebenen
 ' Zeichens (P.flg% = 0) bzw. auf das erste Vorkommen eines
 ' Zeichens, das nicht in der Liste enthalten ist (P.flg% >0).
 ' Wird es gefunden, wird der String bei diesem Zeichen geteilt und
```

```
' die zwei Teile sowie das entsprechende (Trenn-)Zeichen zurückgegeben.
'
' P.flg% = Flag, das angibt, ob auf gleich (0) oder
'          ungleich (>0) geprüft werden soll.
' P.str$ = Zu untersuchender String
' P.sgn$ = String, der das/die (Trenn-)Zeichen enthält
' P.adr% = Pointer auf eine Rückgabe-String-Variable,
'          an die der vordere String-Teil übergeben wird.
' D.adr% = Pointer auf eine Rückgabe-String-Variable,
'          an die der hintere String-Teil übergeben wird.
' F.adr% = Pointer auf eine Rückgabe-String-Variable, an
'          die das gefundene (Trenn-)Zeichen übergeben wird
'
LOCAL p.cnt%
IF LEN(p.str$)                      ! Ziel-String übergeben?
  FOR p.cnt%=LEN(p.str$) DOWNTO 1   ! Alle Ziel-String-Zeichen >--------.
    EXIT IF (INSTR(p.sgn$,MID$(p.str$,p.cnt%,1))=0) AND p.flg%          !
    '                               ! EXIT, wenn Flag > 0 und String-Z. !
    '                               ! im Such-Zeichen-String nicht enth.!
    EXIT IF INSTR(p.sgn$,MID$(p.str$,p.cnt%,1)) AND (p.flg%=0)          !
    '                               ! EXIT, wenn Flag = 0 und String-Z. !
    '                               ! im Such-Zeichen-String enthalten  !
  NEXT p.cnt%                       ! Nächstes Zeichen  <---------------'
  IF p.cnt%<1 OR LEN(p.str$)=1      ! Ziel-String nur ein Zeichen
    '                               ! oder alle Zeichen durchsucht?
    IF (p.cnt%=1 AND p.flg%=0) OR (p.cnt%<1 AND p.flg%>0)
      ' Flag = 0 und Ziel-String ist nur ein Zeichen lang
      ' ODER
      ' Flag > 0 und Ziel-String wurde ohne Erfolg durchsucht?
      *p.adr%=""                    ! Linker String-Anteil ist leer
      *d.adr%=""                    ! Rechter String-Anteil ist leer
      *f.adr%=p.str$                ! Gefundenes Zeichen = Ziel-String
    ELSE                            !
      ' Flag > 0 und Ziel-String ist nur ein Zeichen lang
      ' ODER
      ' Flag = 0 und Ziel-String wurde ohne Erfolg durchsucht !
      *p.adr%=p.str$                ! Linker String-Teil = Ziel-String
      *d.adr%=""                    ! Rechter String-Anteil ist leer
      *f.adr%=""                    ! Kein gefundenes Trennzeichen
    ENDIF
  ELSE                              ! Gleich/Ungleich-Zeichen gefunden!
    IF p.flg%                       ! Flag >0
      ADD p.cnt%,1                  ! Trennposition +1
    ENDIF
    *p.adr%=LEFT$(p.str$,p.cnt%-1)! Linken Teil isolieren
    *d.adr%=RIGHT$(p.str$,LEN(p.str$)-LEN(LEFT$(p.str$,p.cnt%)))
    '                               ! Rechten Teil isolieren
    *f.adr%=RIGHT$(LEFT$(p.str$,p.cnt%)) ! Zeichen isolieren
  ENDIF
ENDIF
RETURN
*****************************************************************************
PROCEDURE xreplc                    ! Steuerzeichenliste ersetzen
  IF rep$>"" AND buf$>""            ! Zeichenliste und Ziel-String okay?
    FOR x.i%=1 TO LEN(rep$)         ! Steuerzeichenliste durchgehen >--.
      @rplc(1,0,buf$,MID$(rep$,x.i%,1),CHR$(255),*buf$)  ! Ersetzen    !
    NEXT x.i%                       ! Nächstes Zeichen <---------------'
  ENDIF
RETURN
```

```
'*************************************************************************
PROCEDURE rplc(r.flg%,pos%,m.str$,s.str$,r.str$,r.adr%)
  ' Setzt an Stelle des gefundenen Such-Strings den angegebenen
  ' Ersatz-String.
  ' =======> Die Prozedur Replc finden Sie unter DATA.
RETURN
'*************************************************************************
PROCEDURE dcolor
  ' Bietet eine komfortable Möglichkeit zur Farbeinstellung
  ' für alle Bildschirmauflösungen.
```

```
  ' Bedienung:
  '     1.        Gewünschtes Farbfeld anklicken.
  '               RGB-Anteile der Farbe werden angezeigt (0 - 7).
  '     2.        Farbanteile bestimmen
  '               Dazu wird mit der linken Maustaste im jeweiligen
  '               Anteilsfeld (+-) der jeweilige Farbanteil um
  '               1 erhöht und mit der rechten Maustaste der Anteil
  '               um 1 vermindert. Änderung wird angezeigt.
  '    3.         Ausgang:
  '               Cancel bzw. <Esc> restauriert die vor der Änderung
  '               gültigen Farben.
  '               Return bzw. <Return> installiert die neue
  '               Farbpalette im System
  '
  ' Wird ein Farbfeld mit der rechten Maustaste angeklickt, wartet die
  ' Routine auf einen weiteren Klick auf ein zweites Farbfeld
  ' (blinkender Rahmen um Feld 1).
  ' Wird das zweite Farbfeld mit der linken Maustaste bestimmt,
  ' werden die beiden Farbregister miteinander vertauscht.
  ' Wird es dagegen mit der rechten Maustaste bestimmt, wird
  ' der Wert des ersten Registers in das zweite Register kopiert.
  '
  ' Die Routine paßt sich selbständig an die aktuelle Auflösung an.
  ' In Hires können nur die Farbregister vertauscht, jedoch nicht
  ' editiert werden.
  ' Von der Routine werden die DEFFILL-, DEFTEXT- und GRAPHMODE-
  ' Attribute verändert. Um die Routine für alle TOS-Versionen
  ' verwendbar zu machen, werden diese Einstellungen nicht
  ' restauriert. Dies muß nach Rückkehr vom Programmierer selbst
  ' erledigt werden.
  ' Bei dieser Prozedur ist programmtechnische Kritik möglich
  ' (z.B. mehrere IF-Abfragen zur Vermeidung von Unterroutinen).
  ' Bedenken Sie bitte dabei, daß es in jeder Hinsicht darum ging,
  ' die Routine absolut unabhängig - und damit selbständig - zu machen.
  '
  LOCAL ci%,cxt%,plane%,breit%,cxl%,cyo%,cxr%,cyu%,bbr,ct%,bx2$,button%
  LOCAL col$,ctx$,bx$,rgb%,key$,i.mem%,cindex%,...
                                 ...ci1%,cbx1%,cby1%,cbx2%,cby2%
  LOCAL rot%,gruen%,blau%,ccolor%,cbk%,...
                                 ...ci2%,cc1%,cc2%,cbr,dbr,dummy$,exflg%
  DIM old_pal%(16),cv%(2,16)
```

```
FOR ci%=0 TO 15                       ! Alle 16 Farbregister >--.
  old_pal%(ci%)=XBIOS(7,ci%,-1) AND &HFFF ! auslesen             |
NEXT ci%                              ! Nächstes Register  <----'
cxt%=2-SGN(XBIOS(4))                  ! X-Auflösungsteiler
cyt%=MIN(2,3-XBIOS(4))                ! Y-Auflösungsteiler
GET 0,0,1,1,dummy$                    ! Dummy-GET
plane%=2^DPEEK(VARPTR(dummy$)+4)      ! GFA-Schlüssel für die Farbebenen
RESTORE convert.table                 ! DATA-Zeiger auf...
convert.table:                        ! ...TOS/GFA-Konvertierungswerte
'                                     !   (siehe unter SETCOLOR)
DATA 0,2,3,6,4,7,5,8,9,10,11,14,12,15,13,1
FOR ci%=0 TO 15                       ! GFA-SETCOLOR-Lowres-Index >----.
  READ ccolor%                        ! Entsprechenden TOS-Index lesen |
  cv%(1,ci%)=ccolor%                  ! GFA-Tabelle belegen            |
  cv%(2,ccolor%)=ci%                  ! TOS-Tabelle belegen            |
NEXT ci%                              ! Nächstes Register <-----------'
IF cxt%=1                             ! Midres oder Hires?
  IF cyt%=2                           ! Midres?
    cv%(1,3)=1                        ! GFA-Index anpassen--. Max. vier
    cv%(2,1)=3                        ! TOS-Index anp. --' Register (0 - 3)
  ELSE                                ! Hires?
    cv%(1,1)=1                        ! GFA-Index anpassen--. Max. zwei
    cv%(2,1)=1                        ! TOS-Index anp. --' Register (0 - 1)
  ENDIF
ENDIF
breit%=400/cxt%                       ! Formularbreite berechnen
cxl%=320/cxt%-breit%/2                ! Linke X-Koordinate
cyo%=160/cyt%                         ! Obere Y-Koordinate
cxr%=320/cxt%+breit%/2                ! Rechte X-Koordinate
cyu%=240/cyt%                         ! Untere Y-Koordinate
GET cxl%,cyo%,cxr%,cyu%,col$          ! Hintergrund puffern
bbr=(breit%-20)/plane%                ! Farb-Button-Breite
cbr=(breit%-20)/5                     ! Wurzel-Button-Breite
dbr=(breit%-20)/15                    ! Klick-Button-Breite
DEFFILL ,0,0                          ! DEFFILL weiß
' ------------------------------------------.
PBOX cxl%,cyo%,cxr%,cyu%                     !
BOX cxl%,cyo%,cxr%,cyu%                      !
BOX cxl%+2,cyo%+2,cxr%-2,cyu%-2              !
FOR ci%=0 TO plane%-1                        !
  DEFFILL ci%,2,8                            !
  PBOX cxl%+11+ci%*bbr,cyo%+12/cyt%,cxl%+8+ci%*bbr+bbr,cyo%+40/cyt%
  BOX cxl%+11+ci%*bbr,cyo%+12/cyt%,cxl%+8+ci%*bbr+bbr,cyo%+40/cyt%
NEXT ci%                                     !
FOR ci%=0 TO 4                               !
  BOX cxl%+10+ci%*cbr,cyo%+48/cyt%,cxl%+10+ci%*cbr+cbr,cyo%+68/cyt%
NEXT ci%                                     !
BOX cxl%+10+4*cbr-1,cyo%+48/cyt%-1,cxl%+11+4*cbr+cbr,cyo%+68/cyt%+1
DEFTEXT 1,,,5/cxt%+cxt%                      !
DEFFILL ,2,8                                 !
RESTORE ctext                                !  Formular
FOR ci%=0 TO 8                               !- zeichnen
  GRAPHMODE 2                                !
  READ ctx$                                  !
  BOX cxl%+9+ci%*dbr,cyo%+48/cyt%,cxl%+9+ci%*dbr+dbr,cyo%+68/cyt%
  TEXT cxl%+12+ci%*dbr+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$
  IF ci%=0 OR ci%=3 OR ci%=6                 !
    GRAPHMODE 3                              !
    PBOX cxl%+9+ci%*dbr,cyo%+48/cyt%,cxl%+9+ci%*dbr+dbr,cyo%+68/cyt%
```

```
  ENDIF                                           !
NEXT ci%                                          !
ctext:                                            !
DATA R,±, ,G,±, ,B,±,
' Möglicherweise ist das in dieser DATA-Zeile    !
' verwendete Sonderzeichen im Buchabdruck        !
' nicht zu erkennen.                             !
' Es handelt sich um drei Data-Sequenzen         !
' zu je drei Zeichen:                            !
' R  + Chr$(241)  + ein Space                    !
' G  + Chr$(241)  + ein Space                    !
' B  + Chr$(241)  + ein Space                    !
TEXT 362/cxt%+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*...
                        ...(2-cyt%),64/cxt%,"Cancel"
TEXT 434/cxt%+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*
                        ...(2-cyt%),64/cxt%,"Return"
GET cxl%+9,cyo%+12/cyt%-2,cxl%+10+bbr,cyo%+40/cyt%+2,bx$
'  Box Cxl%+9,Cyo%+12/Cyt%-2,Cxl%+10+Bbr,Cyo%+40/Cyt%+2
GRAPHMODE 1                                       !
rot%=(XBIOS(7,0,-1) AND &HF00)/256                ! Rot  isolieren
gruen%=(XBIOS(7,0,-1) AND &HF0)/16                ! Grün isolieren
blau%=(XBIOS(7,0,-1) AND &HF)                     ! Blau isolieren
TEXT cxl%+13+2*dbr+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*(2-cyt%),HEX$(rot%)
TEXT cxl%+13+5*dbr+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*
                        ...(2-cyt%),HEX$(gruen%)
TEXT cxl%+13+8*dbr+(2-cxt%)*3,(cyo%+66/cyt%)-cxt%-4*
                        ...(2-cyt%),HEX$(blau%)
' -----------------------------------------'
REPEAT                              ! Hauptschleife >--------------.
  DO                                ! Auswahl-Warteschleife  >----. |
    IF MOUSEK AND MOUSEX>cxl%+9 AND MOUSEX<cxl%+9+11*dbr          ! |
      '                             ! Mausklick innerhalb der Box?| |
      IF MOUSEY>cyo%+48/cyt% AND MOUSEY<cyo%+68/cyt%              ! |
        '                           ! Maus auf RGB-Balken?        | |
        rgb%=INT((MOUSEX-cxl%-9)/dbr) ! Index der gewählten Box   | |
        IF rgb%=1 OR rgb%=4 OR rgb%=7 ! Auf einer Plus/Minus-Box? | |
          IF rgb%=1                 ! Auf R-Box?                  | |
            IF MOUSEK=1             ! Linke Maustaste gedrückt?   | |
              rot%=(rot%+1) MOD 8   ! Rot-Anteil erhöhen          | |
            ELSE                    ! Rechte Maustaste gedrückt?  | |
              DEC rot%              ! Rot-Anteil vermindern       | |
              IF rot%<0             ! Rot kleiner Null?           | |
                rot%=7              ! Dann Rot = 7                | |
              ENDIF                                               ! |
            ENDIF                                                 ! |
            ctx$=HEX$(rot%)                                       ! |
            TEXT cxl%+13+2*dbr+(2-cxt%)*3,...                     ! |
                     ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$       ! |
          ENDIF                                                   ! |
          IF rgb%=4                 ! Auf G-Box?                  | |
            IF MOUSEK=1             ! Linke Maustaste gedrückt?   | |
              gruen%=(gruen%+1) MOD 8 ! Grün-Anteil erhöhen       | |
            ELSE                    ! rechte Maustaste gedrückt?  | |
              DEC gruen%            ! Grün-Anteil vermindern      | |
              IF gruen%<0           ! Grün kleiner Null?          | |
                gruen%=7            ! dann Grün = 7               | |
              ENDIF                                               ! |
            ENDIF                                                 ! |
            ctx$=HEX$(gruen%)                                     ! |
```

```
          TEXT cxl%+13+5*dbr+(2-cxt%)*3,...                          !
                              ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$ !
        ENDIF                                                        !
        IF rgb%=7                    ! Auf B-Box?                    !
          IF MOUSEK=1                ! Linke Maustaste gedrückt?     !
            blau%=(blau%+1) MOD 8    ! Blau-Anteil erhöhen           !
          ELSE                       ! Rechte Maustaste gedrückt?    !
            DEC blau%                ! Blau-Anteil vermindern        !
            IF blau%<0               ! Blau kleiner Null?            !
              blau%=7                ! Dann Blau = 7                 !
            ENDIF                                                    !
          ENDIF                                                      !
          ctx$=HEX$(blau%)                                           !
          TEXT cxl%+13+8*dbr+(2-cxt%)*3,...                          !
                              ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$ !
        ENDIF                                                        !
        VOID XBIOS(7,cv%(2,cindex%),rot%*256+gruen%*16+blau%)        !
        '                            ! Neuen Farbwert initial.       !
        PAUSE 5                      ! Kleine Klickpause             !
      ENDIF                                                          !
    ENDIF                                                            !
  ENDIF                                                              !
  key$=INKEY$                      ! Tastatur abfragen               !
  button%=MOUSEK                   ! Maustasten abfragen             !
  IF MOUSEK AND MOUSEX>cxl%+10+3*cbr AND MOUSEY>cyo%+48/cyt%         !
    IF MOUSEX<cxl%+10+3*cbr+cbr AND MOUSEY<cyo%+68/cyt%              !
      '                            ! Cancel-Button angeklickt?       !
      key$=CHR$(27)                ! <Esc> simulieren                !
    ENDIF                                                            !
  ENDIF                                                              !
  IF MOUSEK AND MOUSEX>cxl%+10+4*cbr AND MOUSEY>cyo%+48/cyt%         !
    IF MOUSEX<cxl%+10+4*cbr+cbr AND MOUSEY<cyo%+68/cyt%              !
      '                            ! Return-Button angeklickt?       !
      key$=CHR$(13)                ! <Return> simulieren             !
    ENDIF                                                            !
  ENDIF                                                              !
  EXIT IF key$=CHR$(13) OR key$=CHR$(27) ! EXIT, wenn <Return>       !
  '                                ! oder <Esc> gedrückt wurde       !
  exflg%=0                         ! Exit-Flag löschen               !
  IF button% AND MOUSEX>cxl%+11 AND MOUSEX<cxr%-11                   !
    IF MOUSEY>cyo%+12/cyt% AND MOUSEY<cyo%+40/cyt%                   !
      exflg%=1                     ! Exit-Flag setzen, ...           !
      '                            ! ...wenn Auswahlbox angeklickt   !
      ' Diese Abbruch-Abfrage ist deshalb so umständlich, um sie     !
      ' im Buch auf Seitenbreite zu bringen.                         !
    ENDIF                                                            !
  ENDIF                                                              !
  EXIT IF exflg%                   ! Abbruch, wenn Exit-Flag gesetzt !
  '                                ! EXIT, wenn Auswahlbox gewählt   !
LOOP                               ! <-------------------------------'
IF button% AND (MOUSEX>cxl%+11) AND (MOUSEX<cxr%-11)                   !
  IF (MOUSEY>cyo%+12/cyt%) AND (MOUSEY<cyo%+40/cyt%)                   !
    '                              ! Eine Auswahlbox angeklickt?       !
    IF button%=1                   ! Mit linker Maustaste?             !
      REPEAT                       ! 1. Warteschleife >-------------.  !
        i.mem%=cindex%             ! alten Farbindex merken         !  !
        REPEAT                     ! 2. Warteschleife >-----------. !  !
          IF MOUSEX>cxl%+11 AND MOUSEX<cxr%-11                    ! !  !
            IF MOUSEY>cyo%+12/cyt% AND MOUSEY<cyo%+40/cyt%        ! !  !
```

```
          '                     ! Maus innerhalb des Formulars?
        cindex%=MAX(0,MIN(plane%-1,INT((MOUSEX-cxl%-9)/bbr)))
          '                     ! Index der gewählten Farbbox
        ENDIF
      ENDIF                     !
    UNTIL MOUSEK=0 OR cindex%<>i.mem% ! EXIT, w. Maustaste
    '                           ! gelöst oder Farbbox-Wechsel <-'
    PUT cxl%+9+i.mem%*bbr,cyo%+12/cyt%-2,bx$ ! Farbbox-Restore
    cbx1%=cxl%+9+cindex%*bbr !-----.
    cby1%=cyo%+12/cyt%-2           !
    cbx2%=cxl%+10+cindex%*bbr+bbr  ! Box-Koordinaten
    cby2%=cyo%+40/cyt%+2     !-----'
    GET cbx1%,cby1%,cbx2%,cby2%,bx$ ! Neue Farbbox puffern
    BOX cbx1%,cby1%,cbx2%,cby2%     ! "Gewählt"-Effekt
    rot%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF00)/256 !Rotanteil
    gruen%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF0)/16 !Grünanteil
    blau%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF)      !Blauanteil
    ctx$=HEX$(rot%)
    TEXT cxl%+13+2*dbr+(2-cxt%)*3,...
                ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$
    ctx$=HEX$(gruen%)
    TEXT cxl%+13+5*dbr+(2-cxt%)*3,...
                   ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$
    ctx$=HEX$(blau%)
    TEXT cxl%+13+8*dbr+(2-cxt%)*3,...
                   ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$
    ' Die einzelnen Farbanteile als Hexadezimalwerte ausgeben
  UNTIL MOUSEK=0                ! Bis die Maustaste gelöst wird <-'
ELSE                            ! Rechte Maustaste gedrückt!
  '                             ! (= Register-Swap oder -Copy)
  ci1%=INT((MOUSEX-cxl%-9)/bbr) ! Index der gewählten Farbbox
  GET cxl%+10,cyo%+12/cyt%-1,cxr%-9,cyo%+44/cyt%+1,bx2$
  '                             ! Hintergrund sichern
  cbx1%=cxl%+10+ci1%*bbr        ! ---.
  cby1%=cyo%+12/cyt%-1             !
  cbx2%=cxl%+9+ci1%*bbr+bbr        ! Box-Koordinaten
  cby2%=cyo%+40/cyt%+1          ! ---'
  BOX cbx1%,cby1%,cbx2%,cby2% ! Rahmen drum
  REPEAT                        ! Warte...
  UNTIL MOUSEK=0                ! ... bis Maustaste gelöst
  DO                            ! Blinkschleife >----------------.
    IF TIMER-ct%>50             ! 1/4 Sekunde vergangen?
      ct%=TIMER                 ! Systemzeit merken
      PUT cxl%+10,cyo%+12/cyt%-1,bx2$ ! Hintergrund-Restore
      GET cxl%+10,cyo%+12/cyt%-1,cxr%-9,cyo%+40/cyt%+1,bx2$
      '                         ! Neuen Hintergrund sichern
      PAUSE 2                   ! Kleine Pause
      BOX cbx1%,cby1%,cbx2%,cby2% ! Wieder Rahmen drum
    ENDIF
    exflg%=0                    ! Exit-Flag löschen
    IF MOUSEK AND MOUSEX>cxl%+11 AND MOUSEX<cxr%-11
      IF MOUSEY>cyo%+12/cyt% AND MOUSEY<cyo%+40/cyt%
        exflg%=1                ! Exit-Flag setzen,...
        '                       ! ...wenn Klick innerhalb der Box
        ' Auch diese Abbruch-Abfrage ist deshalb so
        ' umständlich, um sie auf Seitenbreite zu bringen.
      ENDIF
    ENDIF
  LOOP
```

```
      '                           ! Bis zum nächsten Mausklick    |  |
      '                           ! innerhalb einer Farbbox <-------'  |
      ci2%=INT((MOUSEX-cxl%-9)/bbr) ! Neuer Farbbox-Index             |
      cbx1%=cxl%+10+ci2%*bbr     ! ---.                               |
      cbx2%=cxl%+9+ci2%*bbr+bbr ! ---' Neue Box-X-Koordinaten         |
      BOX cbx1%,cby1%,cbx2%,cby2% ! Noch einmal Rahmen drum           |
      cc1%=XBIOS(7,cv%(2,ci1%),-1)! Farbwert 1.Swap/Copy-Register     |
      cc2%=XBIOS(7,cv%(2,ci2%),-1)! Farbwert 2.Swap/Copy-Register     |
      IF MOUSEK=1                 ! 2. Reg. m. li. Maustaste gewählt? |
        VOID XBIOS(7,cv%(2,ci1%),cc2%) ! Dann 2. Register in das      |
      ENDIF                       ! 1. Register kopieren              |
      VOID XBIOS(7,cv%(2,ci2%),cc1%) ! 1. Register in 2. kopieren     |
      rot%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF00)/256 ! Aktuelle..   |
      gruen%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF0)/16 ! RGB-Werte.   |
      blau%=(XBIOS(7,cv%(2,cindex%),-1) AND &HF)     ! merken.        |
      ctx$=HEX$(rot%)                                                 !
      TEXT cxl%+13+2*dbr+(2-cxt%)*3,...                               |
                        ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$        |
      ctx$=HEX$(gruen%)                                               !
      TEXT cxl%+13+5*dbr+(2-cxt%)*3,...                               |
                        ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$        |
      ctx$=HEX$(blau%)                                                !
      TEXT cxl%+13+8*dbr+(2-cxt%)*3,...                               |
                        ...(cyo%+66/cyt%)-cxt%-4*(2-cyt%),ctx$        |
      ' Die einzelnen Farbanteile als Hexadezimalwerte ausgeben       |
      PAUSE 10                    ! Kleine Pause                      |
      PUT cxl%+10,cyo%+12/cyt%-1,bx2$  ! Hintergrund-Restore          |
    ENDIF                                                             !
  ENDIF                                                               !
ENDIF                                                                 !
UNTIL key$=CHR$(13) OR key$=CHR$(27) ! Bis <Esc> oder <Return> <------'
IF key$=CHR$(27)                  ! <Esc> gedrückt?
  FOR ci%=0 TO 15                 ! Dann alle 16 Register wieder...
    VOID XBIOS(7,ci%,old_pal%(ci%)) ! ... auf alten Wert setzen
  NEXT ci%
ENDIF
PUT cxl%,cyo%,col$                ! Formular-Hintergrund restaurieren
ERASE old_pal%()                  ! Felder...
ERASE cv%()                       ! ...löschen...
RETURN                            ! ...und zurück zum Programm
**********************************************************************
```

24. Anhang

24.1 Der V2.xx-Compiler

Sie kennen C-, Modula- oder Pascal-Compiler? Wenn Sie mit diesem BASIC-Compiler arbeiten, können Sie alles, was Sie über die herkömmliche Arbeisweise eines Compilers wissen (Linking, Bibliotheken, etc.) getrost vergessen. Als Autor fällt es mir schwer, über eine Sache zu schreiben, die in sich so komplex ist und sich trotzdem dermaßen einfach bedienen läßt. Es ist meist leichter, eine Sache zu behandeln, die umfangreiche Probleme aufwirft. Wie Sie sehen werden, ist die Compilerbedienung mit etwas Übung recht einfach zu bewerkstelligen.

Die Komplexität dieses Programms läßt sich zum Beispiel an seinem Umfang erkennen. Der Compiler und das dazugehörige RSC-File nehmen zusammen ganze (ca.) 55 KByte in Anspruch. Gute Compiler anderer Sprachen kommen (inkl. Linker) ungefähr auf den gleichen Umfang. Allerdings muß man dort immer noch ca. 40 - 80 KByte für die bereitzustellenden Libraries (Bibliotheken) mitbedenken. Ich nehme an, daß sich in Zukunft viele Interessenten mit dieser neuen Technik des Compilerbaus auseinanderzusetzen haben werden.

Was ebenfalls fast unglaublich klingt, ist die Tatsache, daß ein .BAS-Programm von ca. 100 KByte in noch nicht einmal drei Minuten komplett geladen, compiliert und als fertiges Maschinenprogramm auf Diskette zurückgeschrieben wird. Von kleinen BASIC-Programmen (bis ca. 10 KByte) braucht man gar nicht zu reden. Diese werden in einer Geschwindigkeit "abgefertigt", daß man noch nicht einmal Zeit hat, an die berühmte Tasse Kaffee zu denken, die man sich bei vielen anderen Compilern in der Zwischenzeit holen und trinken kann. Der Grund für diese "Raserei" ist zum großen Teil eben darin zu suchen, daß dieses Programm eine überaus komplexe Struktur mit kurzen Wegen aufweist und zu einem anderen großen Teil darin, daß es sich hier um einen 100%igen RAM-Compiler handelt. Während des Compiler-Laufs wird also nicht auf den Massenspeicher zugegriffen um evtl. irgendwelche Arbeits-Files unterzubringen. Das zu compilierende Programm kann sich also auch auf einer schreibgeschützten Diskette befinden, ohne daß dieses die Compiler-Arbeit behindern würde.

Wem zudem noch der Komfort einer RAM- oder Hard-Disk zur Verfügung steht, muß aufpassen, daß er nicht dem Compiler-Rausch verfällt. Hier kann man davon ausgehen, daß sich die Zeiten (haupt-

sächlich Lade- u. Speicherzeiten) ca. um 80 Prozent verkürzen. Wer sich schon einmal an anderen Compilern versucht hat, wird sicher haarsträubende Geschichten über die Fehlerbeseitigung an fertigen Programmen berichten können. Programm editieren - compilieren - linken - testen - Programm läuft nicht richtig. Also noch einmal: Programm in Editor laden - editieren - compilieren - .... usw.

Wir besitzen ja nun einen BASIC-Compiler, der vollkompatibel zum Interpreter ist. Also haben wir diese Probleme nicht, da wir uns zur Edition und zum Debugging bequem hinter den Interpreter klemmen und all seine Vorteile nutzen können. Ist das Programm fertig, wird es durch den Super-Compiler gejagt und siehe da - plötzlich befindet sich ein lauffähiges PRG-Programm auf der Diskette. Man startet es durch Anklicken vom Desktop aus und siehe wieder da - es läuft! Es gibt nur wenige Ausnahmen, in welchen das fertige Maschinenprogramm nicht so arbeitet, wie man es erwartet hat.

Eine Möglichkeit, daß das Maschinenprogramm nicht ordnungsgemäß arbeitet, ist dadurch gegeben, daß man die wenigen Compiler-Optionen (siehe unten und unter OPTION) nicht richtig eingesetzt hat. In diesen Fällen wäre unter Umständen ein erneutes Editieren des BAS-Programms notwendig. Das dürfte sich allerdings mit wachsender Erfahrung mehr und mehr erübrigen. Weitere Fehler sind darin zu suchen, daß der Compiler eine völlige andere Arbeitsweise besitzt, als der Interpreter. Daraus ergeben sich auch Unterschiede in der Verwendung einiger Befehle. Diese halten sich allerdings in solch engen Grenzen, daß die Unterschiede im Umgang mit Ihnen nach einiger Zeit in Fleisch und Blut übergegangen sein werden. Weiter unten wird auf diese Problematik noch einmal gesondert eingegangen.

Man kann ohne Übertreibung sagen, daß dieser Compiler Maßstäbe setzt und alle Compiler, die in Zukunft entwickelt werden, werden sich an diesen Maßstäben messen lassen müssen.

24.1.1 Compiler/Interpreter

Was ist eigentlich der Unterschied zwischen einem Compiler und einem Interpreter? Ein Interpreter ist eigentlich ein Dolmetscher, der einerseits den eingegeben Programm-Code Befehl für Befehl in Maschinen-Code übersetzt und andererseits diese der Reihe nach sofort ausführt. Der Programmlauf wird durch den Interpreter kontrolliert. Ist das Programm unterbrochen oder beendet, kehrt das Programm zur Eingabe-Ebene zurück und bietet die Möglichkeit, es sofort wieder

neu zu editieren. Treten also im Programmlauf evtl. logische oder syntaktische Fehler auf, ist es somit leicht möglich, diese zu verfolgen und zu verbessern.

Bei einem Compiler hat man diese Möglichkeiten nicht, da aus dem Programmtext der jeweiligen Sprache ein komplettes Maschinenprogramm erzeugt wird, das dann vom Desktop aus durch einfachen Doppelklick gestartet werden kann. Dieses Programm ist dann absolut unabhängig von einem Übersetzer, da das System bzw. der Prozessor den Maschinen-Code direkt verstehen und verarbeiten kann. Durch Wegfall einer zwischengeschalteten Instanz ergibt sich so eine wesentlich gesteigerte Arbeitsgeschwindigkeit. Bei dem GFA-Compiler bedeutet dies, daß im Vergleich zum Interpreter eine Geschwindigkeitssteigerung von bis zu 2000 Prozent erreicht werden kann. Die Programme laufen also je nach Art und Effektivität des Source-Codes (BASIC-Quellprogrammtext) bis zu 20 mal schneller.

Stellen Sie sich ein normales Fahrrad und einen "normalen" Porsche in Höchstgeschwindigkeit vor. Das wäre das Geschwindigkeitsverhältnis zwischen dem Interpreter- und dem Compiler-Programm. Hier muß allerdings einschränkend gesagt werden, daß diese Steigerung nur unter Optimalbedingungen erreicht wird. Z.B. bei rein arithmetischen Programmen, die nur Integerzahlen verarbeiten und auf Bildschirmausgaben jeglicher Art weitgehend, sowie auf Compiler-Optionen völlig verzichten. Im Normalfall wird sich mit dem Compilat eine drei- bis sechsfache Steigerung erzielen lassen. Um beim obigen Beispiel zu bleiben, wäre dies ein Verhältnis wie zwischen einem guten Moped und einem Porsche. Hier zeigt sich eindringlich der Vorteil, in Programmen soweit es irgend möglich ist, Integervariablen zu verwenden.

24.1.2 Compiler-Bedienung

Falls es noch nicht deutlich geworden ist, soll hier noch einmal die Verfahrensweise im Umgang mit dem Compiler erläutert werden. Nachdem Sie ein Programm auf dem Interpreter erarbeitet, getestet und evtl. Fehler beseitigt haben, speichern Sie es mit der Editor-Funktion Save oder mit dem Befehl SAVE Programmname im Direktmodus auf Diskette oder RAM- bzw. Hard-Disk ab. Verwenden Sie hier bitte weder Save,A im Editor- Menü, noch die Befehle LIST bzw. PSAVE, da der Compiler alle Source-Codes dieser Art ablehnt.

Das Programm befindet sich nun auf dem Massenspeicher. Wenn Sie den Compiler nicht durch EXEC vom Interpreter aus starten, müssen

Sie nun den Interpreter in Richtung Desktop verlassen und dort durch Doppelklick den Compiler starten.

Nachdem Sie das getan haben, erscheint eine Dialogbox. Durch diese Box werden Sie aufgefordert, die eventuell notwendigen Compiler-Optionen zu initialisieren. Falls Sie dieses bereits durch den OPTION-Befehl im Programm-Code getan haben, können Sie diese Eingaben beim Compileraufruf getrost vernachlässigen. Der OPTION-Befehl, sowie die Bedeutung der Dialogbox-Buttons werden weiter unten erklärt.

Anschließend daran klicken Sie das Box-Feld "Compilieren" an oder drücken die <Return>-Taste und wählen in der nun erscheinenden Fileselect-Box das zu compilierende BASIC-Programm aus. Beim ersten Durchlauf wird das BAS-File komplett in den Speicher geladen und auf die Richtigkeit seiner Struktur geprüft. Sollte damit etwas nicht in Ordnung sein, werden Sie aufgefordert, die Mißstände mit dem Interpreter zu korrigieren. Im zweiten Durchlauf wird das Programm nun vollständig compiliert. Und zwar wird das Programm nun abschnittweise noch einmal in den Speicher geladen und schrittweise in ein Maschinenprogramm umgewandelt. Damit dieser Vorgang nicht langweilig wird, wird während des Compilierens der Bildschirm im Wechsel mit schwarzen und weißen Linien gefüllt.

Ist das Compilat fertiggestellt, erscheint erneut eine Fileselect-Box mit der Überschrift "Schreibe PRG". Der Name Ihres BASIC-Programms ist bereits rechts in der Box unter Auswahl mit der Extension .PRG eingetragen. Wollen Sie den Namen beibehalten, klicken Sie einfach OK an. Wenn nicht, können Sie mit den üblichen Editier-Funktionen <Backspace>, <Delete>, <Escape> und den Cursor-Tasten oder durch Anklicken eines Namens im Fenster den Namen ändern. Hier ist zu beachten, daß der Compiler kein Backup-File anlegt, wenn man einen Programmnamen angibt, der bereits auf der Diskette existiert. Das schon bestehende Programm wird gnadenlos überschrieben. Achten Sie außerdem darauf, daß genügend Platz auf der Diskette ist, auf welche Sie das fertige Programm schreiben wollen. Der Compiler verfügt leider nicht über die Fähigkeit, dies zuvor festzustellen und evtl. eine Warnung auszugeben. Ist nicht genügend Platz, erscheint eine Meldung, daß ein Schreibfehler aufgetreten ist und der Compiler kehrt zur Eingabebox zurück. Das heißt, daß das Programm vollständig neu compiliert werden muß. Wechseln Sie also gegebenfalls vorher die Diskette.

Nachdem Sie nun OK gewählt haben, wird das PRG-Maschinenprogramm auf dem Massenspeicher (Diskette, RAM- oder Hard-Disk) abgelegt. Ist das geschehen, meldet sich die Compiler-Dialogbox wieder und Sie können weitere Programme compilieren oder durch "Abbruch" den Compiler verlassen. Auf der Diskette finden Sie nun ein typisches PRG-Programmsymbol mit dem Namen Ihres Programms darunter. Dieses Symbol können Sie nun mit Doppelklick anwählen und so **Ihr** Maschinenprogramm starten. Mehr wäre zur Compilerbedienung eigentlich nicht zu sagen, wenn da nicht die Optionen wären.

24.1.3 Compiler-Optionen

Bei der Befehlsbeschreibung zu OPTION wurden die Erläuterungen relativ kurz gehalten. Da eine nähere Beschreibung hier den entsprechenden Rahmen erhält, wurde sie hierher verlegt.

Mit dem Befehl OPTION können dem Compiler Anweisungen übermittelt werden, wie er bestimmte Zustände verarbeiten soll, bzw. wie Ihr compiliertes Programm später auf diese Zustände reagieren soll.

OPTION "Ux"
Der erste Zustand, der zu behandeln ist, tritt dann ein, wenn während des Laufs des compilierten Programms die (im Interpreter übliche) Abbruchfunktion <Control><Shift><Alternate> benutzt wird. Das U steht oben also für "Unterbrechung". Durch Angabe einer Zahl von 0 bis 3 für x (z.B. U2) kann auf vier verschiedene Arten auf den Unterbrechungsversuch reagiert werden.

OPTION "U0"
Die Abbruchfunktion wird nicht eingebaut. D.h., daß im compilierten Programm keine Unterbrechung durch Betätigung der genannten Breaktasten möglich ist. Eine Programmverzweigung durch ON BREAK GOSUB, die Sie evtl. im Programm vorgesehen haben, ist dadurch allerdings auch nicht mehr möglich. Grundätzlich hat dieser Befehl dieselbe Wirkung im Compilat, wie der Befehl ON BREAK CONT im Interpreter. Der Unterschied ist hier der, daß die notwenigen Routinen zur Behandlung der Abbruchfunktion nicht in das Compilat eingebunden werden und das fertige Programm dadurch etwas kürzer wird. Haben Sie diesen Befehl im Programm nicht verwandt, können Sie das nachholen, indem Sie in der Compilerbox den Button "Stoppen Nie" anklicken.

OPTION "U1"
Es wird genau an der Programmstelle, an welcher dieser OPTION-Befehl auftaucht, **einmal** überprüft, ob die Break-Tastenkombination gedrückt wird. Sie können diese Abbruchbedingung also gezielt an den von Ihnen für wichtig gehaltenen Stellen einsetzen. In diesem Fall ist auch eine Programmverzweigung durch ON BREAK GOSUB möglich. Da diese Anweisung beim Compiler-Aufruf nicht eingesetzt werden kann, ist hierfür auch kein entsprechender Button in der Box vorgesehen.

OPTION "U2"
Mit dieser Option wird der Compiler angewiesen, **vor** jedem Schleifen-Wendepunkt (also LOOP, UNTIL, WEND und NEXT), sowie **vor** jedem GOTO-Befehl **eine** Abfrage einzubauen. Da wohl der größte Teil vieler Programme sich in irgendeiner Schleife die Zeit vertreibt, dürfte diese Option wohl in den meisten Fällen ausreichen. Auch hier kann durch ON BREAK GOSUB eine Unterbrechungsprozedur bestimmt werden. Statt der Verwendung des Befehls im Programm können Sie hierfür auch den Button "Stoppen Schleife" in der Compiler-Box verwenden.

OPTION "U3"
Wer ganz sicher gehen will, daß sein Programm zu jedem beliebigen Zeitpunkt durch die Break-Tastenkombination unterbrochen werden kann, muß diese Option verwenden, da in diesem Fall **nach** Ausführung **jedes** BASIC-Befehls einmal die Abbruchfunktion überprüft wird. Einleuchtenderweise wird dadurch das Programm natürlich erheblich langsamer, weil es mehr arbeiten muß. Wie bei "U0" und "U2" können Sie auch hier die Einstellung in der Dialogbox vornehmen, indem Sie den Button "Stoppen Immer" anklicken. Die Unterbrechungsoptionen haben noch eine eine weitere Funktion zur Feststellung der Sprungmarken bei RESUME (siehe unter 24.1.4 "Allgemeines").

OPTION "T-" / OPTION "T+"
Ein Unterschied in der Arbeitsweise zwischen Interpreter und Compiler besteht darin, daß der Interpreter Integerwerte ausschließlich im festgelegten Integerbereich (-2147483648 bis 2147483647) verarbeiten kann. Werden diese Bereiche über- bzw. unterschritten, wird eine entsprechende Fehlermeldung ausgegeben. Der Interpreter kontrolliert also die Einhaltung dieser Grenzen. Nicht so der Compiler.

Im compilierten Programm werden bestimmte Befehle der Integerarithmetik (INC, DEC) direkt als einzelne Maschinenbefehle ausgeführt. D.h. also, daß die Einhaltung des Integerbereichs nicht kontrolliert wird. Beinhaltet eine Integervariable den Wert 2147483647 (2^31-1) und wird anschließend z.B. durch INC um 1 erhöht, fällt der Variablenwert in den Minusbereich (-2147483648) und würde durch weiter INC-Befehle wieder gegen Null streben. Das würde natürlich gegebenenfalls die Berechnungsergebnisse verfälschen. Um nun in den seltenen Fällen, in welchen mit derart hohen Werten hantiert wird, kontrollieren zu können, ob sich der Wert unter oder über die Integergrenzen hinausbewegt, gibt es diese Option.

OPTION "T-"

Der eben beschriebene Sonderfall wird **nicht** kontrolliert und es wird keine entsprechende Routine eingebaut. Bei Überlauf bewegen sich die Werte also in den Minusbereich und umgekehrt. Klicken Sie in der Compiler-Box "Trapv -" an, erreichen Sie damit denselben Effekt wie mit der Programm-OPTION "T-".

OPTION "T+"

Es wird eine Kontrollroutine eingefügt, die in den entsprechenden Grenzfällen überprüft, ob die Integergrenzen eingehalten werden. Tritt ein Überlauf ein, wird je nachdem, ob die B+-Option eingeschaltet ist oder nicht, entweder die Fehlermeldung "# 107" ausgegeben oder das Programm verabschiedet sich mit den (in GFA-BASIC schon fast vergessenen) 7 Bomben (TRAPV-Interrupt). Wird der Minusbereich dagegen unterschritten, wird das Programm abgebrochen. Da sich die zu errechnenden Integerwerte jedoch meist in wesentlich engeren Grenzen halten werden, ist diese Option nur in seltenen Fällen angebracht. Haben Sie diese Option nicht im Programm verfügt, können Sie sie durch Anklicken des Buttons "Trapv +" in der Compiler-Box dem Compilat einfügen lassen.

OPTION "E+" / OPTION "E-"

Sie kennen die komfortablen Fehlermeldungen des Interpreters. Jede dieser Fehlermeldungen legt einen Code in der reservierten Variablen ERR ab.

Dieser Code kann durch z.B. PRINT ERR ermittelt werden. Sie erhalten dann eine Fehlernummer. In compilierten Programmen werden ebenfalls Fehlermeldungen ausgegeben, sobald ein Fehler auftritt.

OPTION "E-"
Es wird eine Fehlermeldung ausgegeben, welche die Fehlernummer (siehe ERR) sowie den Hinweis "PC>$00XXXXXX" enthält. Um feststellen zu können, um welchen Fehler es sich handelt, müssen Sie also in der Liste der Fehlermeldungen nachschauen. Statt der Verfügung im Programmtext können Sie auch den Button "Errors -" in der Compiler-Box wählen.

OPTION "E+"
Wollen Sie nicht jedesmal in der Fehlerliste nachschauen müssen, können Sie diese Option verwenden oder in der Compilerbox den Button "Errors +" anklicken. Es werden nun auch im compilierten Programm ähnlich ausführliche Fehlermeldungen als Text ausgegeben, wie wir sie vom Interpreter her gewohnt sind. Gerade bei der Programmentwicklung ist diese Möglichkeit sehr von Nutzen, da es die Fehlersuche wesentlich erleichtert. Zusätzlich erscheint wieder der Hinweis "PC>$00XXXXXX". Dieser Hinweis zeigt Ihnen den Stand des System-Programmzählers (PC = Program-Counter) zu dem Zeitpunkt, an welchem der Fehler aufgetreten ist. Der Programm-Counter zeigt in der Regel auf die Speicheradresse, die den Maschinenbefehl beherbergt, der den Fehler auslöste.

OPTION "B-" / OPTION "B-"
Wer schon längere Zeit mit GFA-BASIC arbeitet, wird sie schon fast vergessen haben: die Bomben. Ein Bomben-Error tritt immer dann auf, wenn versucht wird, etwas auszuführen, was im Sytem nicht definiert ist (z.B. Division durch Null, unvorbereiteter Zugriff auf reservierte Supervisor-Bereiche, ungültige Opcodes, Word- und Longword-Zugriffe auf ungerade Adressen etc.). Es gibt also vom System her keine Routinen, die darauf spezialisiert sind, diese Errors sinnvoll abzufangen. Darin liegt auch das Geheimnis der Error-Meldungen des Interpreters selbst bei Bomben-Errors. Man nehme den entsprechenden Exception-Vektor und biege ihn auf eine selbstdefinierte Routine.

Schon gibt es keine Bomben mehr, da die Fehler von einer Interpreter-Routine so abgefangen werden, daß statt der Bomben eine harmlose Fehlermeldung auf dem Bildschirm erscheint. Ganz so einfach ist das nun zwar doch nicht, weil die wichtigen Registerinhalte gerettet und restauriert werden müssen, aber im Prinzip ist dies die Technik der Exception-Verarbeitung (siehe auch MONITOR).

Sie haben nun mit dieser Option die Wahl, in das Compilat die entsprechenden Abfangroutinen einbauen zu lassen oder nicht.

OPTION "B-"
Beim Auftreten von Bombenfehlern (Fehlernummern 102 - 109) werden die entsprechenden Bomben auf dem Bildschirm angezeigt. Dabei bedeutet 2 Bomben = Fehlernummer 102, 3 Bomben = Fehlernummer 103 usw. Dieser klassische "Absturz" hat zur Folge, daß sicherheitshalber das gesamte System neu initialisiert werden muß (Reset), um eine ordnungsgemäße Arbeit des Systems zu gewährleisten. Unter Umständen können die Register bei 2 oder 3 Bomben noch intakt sein, eine Garantie gibt es dafür allerdings nicht. Haben Sie im Programm durch ON ERROR GOSUB eine Routine zur Fehlerbehandlung vorgesehen, wird diese **nicht** angesprungen. Wahlweise können Sie statt der Programmverfügung auch hier den Button "Bomben -" in der Compiler-Box wählen.

OPTION "B+"
Wollen Sie, daß auch Bomben-Errors abgefangen werden und eine entsprechende Fehlermeldung erscheint bzw. zu einer von Ihnen definierten Fehler-Routine verzweigt wird (ON ERROR GOSUB), müssen Sie diese Option im Programmtext übergeben. Wahlweise kann auch der Compiler-Box-Button "Bomben +" gewählt werden. Es wird eine Bombenabfang-Routine in das Compilat eingebaut und ab jetzt arbeitet Ihr fertiges Programm bezüglich der Bombenfehler genauso, wie Sie es vom Interpreter kennen.

24.1.4 Allgemeines

Prioritätenfolge

Bei allen Optionen gilt, daß immer die Option als gültig angenommen wird, auf welche der Compiler zuletzt trifft. Dabei gilt die Dialogboxeinstellung als die erste Instanz. Folgen also im Programm Einstellungen, die sich mit der Einstellung in der Box widersprechen, hat die im Programm übergebene OPTION den Vorrang. Das gleiche gilt für verschiedene Optionen der gleichen Art (U, T, E oder B), die im Programmtext integriert sind. Trifft also der Compiler auf eine U3-Option und Sie haben dementsprechend eine ON BREAK GOSUB-Prozedur definiert, nützt dies wenig, wenn Sie später als letzte Option im Programm die U0-Option angeben. Dadurch wird dem Compiler ab-

schließend gesagt, daß die Abbruchtasten-Funktion nicht eingebaut werden soll. In diesem Fall ist überall dort im Programm mit Fehlverhalten zu rechnen, wo trotzdem diese Funktion abgefragt werden soll, also U1, U2 oder U3 eingesetzt wurde.

Erweiterungen

Das 30. Byte jedes compilierten Programms entscheidet darüber, ob bei Programmstart der Bildschirm gelöscht wird. Wird dieses 30. Byte auf Null gesetzt, kann dadurch das Löschen der Screen unterdrückt werden.

```
Open "U",#1,"Programmname.PRG"
Seek #1,30
Out #1,0
Close
```

Haben Sie es sich anders überlegt und möchten, daß der Bildschirm doch wieder gelöscht wird, ersetzen Sie in dem kleinen Listing oben die Zeile OUT #1,0 durch OUT #1,27. Vorausgesetzt ist dabei, daß das 31ste Byte des Programms nicht verändert wurde, bzw. - falls doch - wieder auf den Wert 69 gesetzt wird.

Die Bytes 30 - 38 eines GFA-Compilats sind für einen String reserviert, der zum Programmstart ausgeführt wird. Und zwar handelt es sich dabei um TOS-Escape-Sequenzen (siehe PRINT). Sie können diese 9 Byte also verwenden, um eigene Sequenzen zu bestimmen, die zu Beginn des Programms ausgeführt werden sollen. Ein solcher String beinhaltet dann aufeinanderfolgend im Wechsel den Escape-Wert 27 und den ASCII-Wert des Zeichens, das die entsprechende Sequenz repräsentiert und/oder die ASCII-Werte der gewünschten Steuerzeichen.

Ein solcher String könnte folgendermaßen aussehen:

```
A$=Chr$(27)+"f"+Chr$(27)+"E"+Chr$(27)+"p"+Chr$(7)+Chr$(0)+Chr$(0)
Open "U",#1,"Programmname.PRG"
Seek #1,30
Print #1;A$;
Close
```

Unter der Befehlsbeschreibung zu PRINT finden Sie die Escape-Sequenzen und die oben verwendeten Buchstaben wieder. Wichtig ist bei Übergabe eines derartigen Strings, daß das letzte Zeichen ein Nullzeichen ist. Sie finden außer den Escape-Definitionen oben auch das Steuerzeichen Chr$(7) (BEL). Wird dieses Steuerzeichen ausgeführt, erklingt die Glocke.

Mauszeiger ein-/ausschalten

Bei den ersten Versuchen, mit dem Compiler lauffähige Programme zu erzeugen, wird Ihnen in manchen Programmen auffallen, daß der Mauzeiger nirgends zu finden ist. Der Grund dafür ist, daß der Compiler den Mauszeiger nicht generell nach jedem Befehl wieder einschaltet. Der Interpreter sorgt permanent dafür, daß der Zeiger ständig sichtbar bleibt (sofern HIDEM nicht aktiv ist). Man braucht sich um ihn also keine Gedanken zu machen.

Das hat allerdings auch den Nachteil, daß er innerhalb von Schleifen häufig unruhig flackert. Wer sich in anderen BASIC-Dialekten schon einmal mit der Erzeugung von Grafikausgaben (PBOX, CIRCLE etc.) durch GEM-Routinen beschäftigt hat, wird wissen, daß dazu jedesmal der Mauszeiger vorher ausgeschaltet werden muß, damit an der Stelle, wo der Zeiger steht, kein "Loch" im Hintergrund entsteht. Dieses Abschalten erfolgt beim Interpreter, ebenso wie bei compilierten Programmen automatisch. Der Unterschied besteht darin, daß das compilierte Programm ihn erst wieder einschaltet, wenn entweder der Befehl SHOWM auftaucht, eine Mausabfrage erfolgt, die AES aufgerufen werden (GEMSYS...) oder eine Ux-Option verfügt wird.

Würde der Befehl zum Einschalten des Mauszeigers wie im Interpreter nach jeder Mausbewegung oder nach jedem Befehl ausgeführt werden, wären die compilierten Programme dadurch erheblich länger und langsamer.

Nicht verwendbare Befehle

Folgende BASIC-Befehle können vom Compiler nicht verarbeitet werde, da sie sich ausschließlich auf die Arbeit mit dem Interpreter auswirken:

```
LIST    LLIST   TRON    TROFF   DEFLIST
SAVE    PSAVE   LOAD    STOP    CONT
```

Trifft der Compiler während seiner Arbeit trotzdem auf einen dieser Befehle, wird der Alert-Box-Text "Fehler beim Compilieren" und der Name des nicht auführbaren Befehls ausgegeben und man kann sich dann durch "Weiter" oder "Abbruch" entscheiden, ob der Compiler seine Arbeit abbrechen soll oder ob der Befehl beim Compilieren ignoriert werden soll.

RESUME-Ausführung

Der Interpreter verfügt während eines Programmlaufs ständig über die genauen Adressen jedes einzelnen Befehls. Es ist also recht unproblematisch mit dem RESUME-Befehl nach Ausführung einer Fehlerbehandlungsroutine zu der Stelle zurückzukehren, an welcher der Fehler aufgetreten ist. Dieser Vorgang gestaltet sich bei compilierten Programmen wesentlich anders. Um feststellen zu können, wo der Fehler aufgetreten ist, muß die Adresse des Befehls, der den Fehler ausgelöst hat, feststehen. Bei Verwendung von RESUME Label ist dies gegeben, da das Label intern für eine bestimmte Adresse steht. Will man allerdings RESUME Next (nächsten Befehl ausführen) oder RESUME allein (fehlerhaften Befehl nochmal ausführen) verwenden, muß man bei compilierten Programmen den OPTION Ux-Befehl verwenden. Dieser Befehl hat außer der Bereitstellung von Unterbrechungsmöglichkeiten auch noch die Funktion, als Sprungadresse für RESUME und RESUME Next zu dienen. Bei Ausführung eines OPTION Ux-Befehls wird nämlich gleichzeitig der aktuelle Programmzähler abgelegt, so daß bei Auftreten eines Fehlers das Programm weiß, an welcher Stelle es durch den Fehler unterbrochen wurde.

Das stimmt bei Verwendung von U1 und U2 natürlich meist nicht exakt, da durch RESUME dann zu der Stelle zurückgekehrt wird, wo der letzte OPTION Ux-Befehl verwendet wurde und nicht exakt dorthin, wo der Fehler aufgetreten ist. Da man glücklicherweise in den meisten Fällen relativ genau weiß, an welcher Programmstelle evtl. ein Fehler eintreten könnte, braucht man also nur direkt vor dieser Stelle OPTION U1 verfügen, zu welcher dann durch RESUME bzw. RESUME Next zurückgekehrt werden kann. Das hat natürlich dann auch zur Folge, daß das Programm auch durch die Break-Tastenfunktion unterbrochen werden kann. Wollen Sie eine Unterbrechung hierdurch vermeiden, setzen Sie am Programmanfang den Befehl ON BREAK CONT.

Anderenfalls können Sie eine Break-Behandlungsroutine bereitstellen, die dann durch ON BREAK GOSUB angesprungen und in welcher auf die Unterbrechung gezielt reagiert werden kann. Dieses etwas umständlich erscheinende Verfahren ist notwendig, da der Einbau einer ständigen Speicherung des Programmzählers in das Compilat mit erheblichem Geschwindigkeitsverlust und einem größeren Programmumfang bezahlt werden müßte.

Beispiel:

```
Option "U0"
Option "T+"
On break gosub Abbruch
On error gosub Fehler
A%=2^31-100
Do
  Option "U1"
  Print A%
  Inc A%
  Exit if Flag%=1
Loop
Print "Schleife verlassen!  A%: ";A%;"  (Taste drücken)"
Void Inp(2)
Quit
Procedure Abbruch
  Alert 2,"Programmende?",1,"OKAY|NEIN",Back%
  If Back%=1
    Quit
  Endif
Return
Procedure Fehler
  Alert 1,"Integerbereich| max. 2147483647",1,"OKAY",Back%
  Flag%=1
  Resume Next
Return
```

CHAIN-Ausführung

Der CHAIN-Befehl wird vom Compiler anders ausgeführt als vom Interpreter. Im Interpreter bezieht sich dieser Befehl auf ein nachzuladendes und vom Interpreter auszuführendes BAS-Programm. Da dies natürlich bei einem compilierten Programm nicht möglich ist, wird hier durch CHAIN ein beliebiges, direkt ausführbares .PRG-, .TOS- oder .TTP-Programm angesprochen. Vor dessen Ausführung wird der angegebene Programmname an die AES-Funktion SHEL_WRITE() (siehe dort) übergeben und das aufrufende Programm beendet. Es ist danach also nicht mehr aufrufbar und der belegte Speicherplatz wird wieder freigegeben. Die zweite Hälfte der Basepage (Bytes 129 - 256) wird durch die AES-Funktion SHEL_WRITE() als Kommandozeile an das aufgerufene Programm übergeben. So ist es möglich, diese 128 Bytes dazu zu verwenden, Informationen, Variableninhalte etc. an das aufgerufene Programm zu übergeben.

Beispiel:

```
** Programm 1 **
A$="Strings werden durch sh_write in Großschrift übergeben!"
Poke Basepage+128,Len(A$)
Bmove Varptr(A$),Basepage+129,Len(A$)
```

```
Print "PROGRAMM 1"
Chain "Prog2.prg"
** Programm 2 **
L.en%=Peek(Basepage+128)
A$=Space$(L.en%)
Bmove Basepage+129,Varptr(A$),L.en%
Print "PROGRAMM 2"
Print "String-Variableninhalt aus Programm 1:"
Print Chr$(10);''A$'';Chr$(13);Chr$(10);Chr$(10);"(Taste drücken)"
Void Inp(2)
```

Speichern Sie die beiden Beispielprogramme als BAS-Files ab. Das zweite Programm muß dabei den Namen PROG2.BAS tragen. Compilieren Sie anschließend beide Programme und starten Sie dann Programm 1. Nach kurzer Zeit wird Programm 1 abgebrochen und Programm 2 gestartet. Anhand des übergebenen Strings kann man sehen, daß es problemlos möglich ist, Informationen zwischen den Programmen auszutauschen. So können z.B. Dateinamen übergeben werden, die dann vom aufgerufenen Programm geladen oder bearbeitet werden. Der Unterschied zu dem Befehl EXEC besteht hauptsächlich darin, daß hier eben die Basepage durch die genannte AES-Funktion übergeben und das aufrufende Programm aus dem Speicher gelöscht wird. Wird dagegen ein Programm durch EXEC aufgerufen, wird die Basepage nicht übergeben und nach Abschluß des aufgerufenen Programms kehrt das System nicht grundsätzlich zum Desktop, sondern zum aufrufenden Programm zurück.

Fileselect-Box-Speicher

Der Aufruf einer FILESELECT-Box gestaltet sich im Interpreter erfreulich problemlos. Von vielen wird vielleicht garnicht bedacht, daß bei jedem Aufruf der Hintergrund (also das aktuelle Bild) zwischengespeichert werden muß, um damit hinterher die Box wieder zu überlagern. Der V2.xx-Interpreter verfügt über zwei Bildspeicher, wovon der Editorbildschirm dazu verwendet wird, bei FILESELECT-Aufrufen den Hintergrund zu retten. Da es bei compilierten Programmen keinen Editor gibt, gibt es natürlich auch keinen Editorbildschirm. Aus diesem Grund muß sichergestellt werden, daß das Programm zum Zeitpunkt des Aufrufs einer FILESELECT-Box über ca. 32500 Byte freien Speicher verfügt.

Dies kann am besten erreicht werden, wenn man den Programmspeicher vor Aufruf der Box durch RESERVE auf diese ca. 32500 Byte plus einer überschlägig angenommenen Größe des benötigten Variablenspeichers festlegt. Die Box verwendet dann den restlichen Variablenbereich zum Speichern der aktuellen Screen und restauriert den

Bildschirm anschließend wieder damit. Wenn Sie keine weiteren Programme in den Speicher laden wollen (siehe EXEC) können Sie auch ganz sicher gehen, indem Sie am Programmanfang einfach RESERVE FRE(0) einsetzen und damit den gesamtem freien Speicher für das aktuelle Programm reservieren.

Sonstiges

Haben Sie alle Optionen bereits im Programm verfügt oder wollen Sie die Optionen U0, T-, E- und B- einsetzen, müssen Sie in der Compiler-Box keinen der Buttons bedienen, da diese bereits voreingestellt sind. Sie erscheinen bei Compiler-Aufruf schwarz unterlegt.

Wenn Sie bei Eingabe der Programm-OPTIONen zwischen dem Befehl und der verfügten Option ein Leerzeichen lassen, können Sie die Anführungszeichen vernachlässigen, sie werden vom Interpreter automatisch gesetzt (z.B. Opt u1). Der Optionsbuchstabe kann wahlweise auch kleingeschrieben sein. Jede Option, die den Komfort des Programms steigert (U1 - U3, T+, E+, B+) vermehrt den Programmumfang und vermindert die Laufgeschwindigkeit des Programms.

Bei Programmen, die den Bildschirmbereich des Editors zur Speicherung eigener Daten verwenden, können Fehlfunktionen im compilierten Programm auftreten. Eine Null-durch-Null-Division (0/0) ergibt nicht, wie beim Interpreter, die Fehlermeldung "Division durch Null", sondern der Compiler erkennt, daß diese Teilung den Wert 1 ergibt und liefert diesen Wert als Ergebnis der Teilung an das Programm (PRINT 0/0 => ergibt 1). Es ist kaum vorstellbar, daß dieser Fall eintritt, aber sicher ist sicher.

Anders als im Interpreter wird in GFA-Compilaten die REPEAT..UNTIL-Schleife mit einem Integerzähler schneller ausgeführt, als eine entsprechende Integer-FOR..NEXT-Schleife.

Beispiel:

```
Repeat              |  ist        |>  For Count%=1 To 1000
  Inc Count%        |  schneller  |>  Next Count%
Until Count%=1000   |  als        |
```

24.2 Variablenorganisation/Typen

Variablennamen ohne Kennung bzw. mit der Kennung # (bei V3.0-DEFLIST 2/3) werden als **Fließkomma**-Variablen (auch **Real**-Variablen genannt) interpretiert. Dieser Variablenyp benötigt zu seiner Speicherung 6 Byte (V2.xx), bzw. 8 Byte (V3.0) Speicherplatz. Im Dezimalbereich kann so eine Genauigkeit von bis zu maximal 11 Stellen (V2.xx) bzw. 13 Stellen (V3.0) eingehalten werden. Nimmt der ganzzahlige Anteil mehr als 11 (bzw. 13) Stellen ein, wird der Variablenwert in das Exponentialformat konvertiert. In diesem Format können dann Werte mit einem ganzzahligen Anteil von bis zu 154 Stellen (V2.xx) bzw. 308 Stellen (V3.0) erfaßt werden. Größter darstellbarer Wert = 1.0E+154 (V2xx) bzw. 3.595386269725E+308 (V3.0).

Variablennamen mit der Kennung (Postfix) % (z.B. Var%) gelten als **4-Byte-Integer**-Variablen. Jeder diesem Variablentyp zugeordnete Wert wird auf seinen ganzzahligen Anteil reduziert. D.h., evtl. auftretenden Nachkommastellen werden "integriert". Zu seiner Speicherung benötigt dieser Typ 4 Byte Speicherplatz (+ Zeichenanzahl des Namens). Es können Werte im Bereich von -2147483648 (= -2^31) bis +2147483647 (= 2^31-1) verarbeitet werden. Werte außerhalb dieses Bereichs werden als Überlauf angesehen.

Variablennamen mit der Kennung | (z.B. Var|) gelten als vorzeichenlose **1-Byte-Integer**-Variablen (-> nur V3.0). Diesem Typ zugeordnete Werte werden auf ihren ganzzahligen Anteil reduziert. Evtl. auftretenden Nachkommastellen werden also auch hier "integriert". Zu seiner Speicherung wird 1 Byte Speicherplatz benötigt (+ Zeichenanzahl des Namens). Es können Werte im Bereich von 0 bis 255 hiermit verarbeitet werden. Werte außerhalb dieses Bereichs werden als Überlauf angesehen.

Variablennamen mit der Kennung & (z.B. Var&) gelten als **2-Byte-Integer**-Variablen (-> nur V3.0). Auch diesem Typ zugeordnete Werte werden auf ihren ganzzahligen Anteil reduziert (integriert). Zu seiner Speicherung benötigt dieser Typ 2 Byte Speicherplatz (+ Zeichenanzahl des Namens). Es können Werte im Bereich von -32768 (= -2^15) bis +32767 (= 2^15-1) verarbeitet werden. Werte außerhalb dieses Bereichs werden als Überlauf angesehen.

Variablennamen mit der Kennung ! (z.B. Var!) sind **Boole**-Variablen. Dieser Variablentyp kann ausschließlich die Werte 0 (= False) und -1 (<> 0 = True) annehmen. Es werden zu seiner Speicherung 2 Byte benötigt (+ Zeichenanzahl des Namens). Im Gegensatz zu anderen Ty-

pen hat diese Variable innerhalb von Feldern jedoch einen anderen Speicherbedarf als den einer Einzelvariablen. In Feldern benötigt sie nur 1 Bit (!) je Element.

Variablennamen mit der Kennung $ (z.B. Var$) werden als **Text-Variablen** (**String**-Variablen) interpretiert. Für diesen Variablentyp wird jeweils ein Descriptor von 6 Byte eingerichtet. In diesem Descriptor befindet sich in den ersten vier Byte als Longword die Adresse des ersten Zeichens und in den nächsten zwei Byte die Länge des Strings.

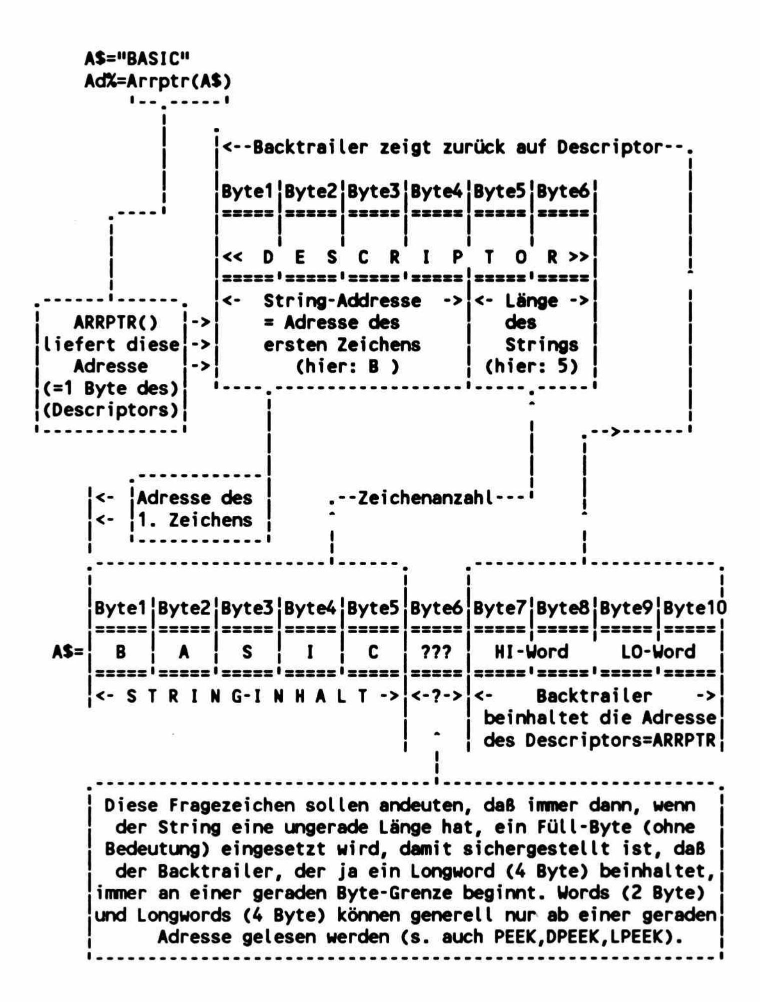
Der String-Descriptor ist über den Befehl ARRPTR(Var$) zu ermitteln. Jedem String, der in einer Variablen gespeichert wird, wird ein 4 Byte langer Backtrailer (Anhänger) zugeordnet. Dieser beinhaltet die Anfangsadresse (Longword) des jeweils zugehörigen Descriptors. Hat der String eine ungerade Länge, wird zwischen String-Ende und Backtrailer ein Füll-Byte gesetzt, damit das Backtrailer-Longword auf jeden Fall eine gerade Adresse erhält.

Benötigter Speicherplatz eines Strings:

```
          6 Byte für Descriptor
        + 4 Byte für Backtrailer
evtl.   + 1 Füll-Byte
        + n Bytes (n = String-Länge)
        + n Bytes (n = Zeichenanzahl des Namens)
```

Der String-Descriptor und der Backtrailer weisen quasi durch ihren Inhalt aufeinander. Dies gewährleistet, daß der ständige Aufräumungsprozess der Garbage-Collection nicht mal "ausnahmsweise" ein Zeichen "verliert". Mancher würde vielleicht sagen: "Das kann ja mal passieren. Ist nicht so schlimm!". Hier darf. es nicht passieren. Jedes Byte, daß verloren ginge, hätte unter Umständen katastrophale Folgen.

Der Aufbau des Descriptors und des Strings mit seinem Backtrailer sieht schematisch so aus: Der String soll einfach "BASIC" heißen:



Die Inhalte werden nun ermittelt, indem mit LPEEK und DPEEK (bzw. CARD{} und LONG{}) die entsprechenden Words und Longwords gelesen werden.

Mit jedem der oben angeführten Variablentypen lassen sich auch ein- oder mehrdimensionale **Felder** (Arrays) bilden. Zur Bestimmung der Adressen der einzelnen Feldelemente und ihrer Größen (bei String-Feldern), sowie zur Ermittlung der Dimensionsanzahl und deren einzelnen Elementmengen kann ebenfalls der Befehl ARRPTR (siehe dort) verwendet werden, der einen Variablen-Descriptor liefert.

Die Möglichkeit, eine Garbage-Collection durchzuführen bzw. zu veranlassen, verlangt bei Feld- und String-Variablen (deren Länge variabel sein kann) eine Information darüber, wo die entsprechenden Einträge zu finden sind. Deshalb ist es nötig, eine solche Informationseinheit irgendwo im Speicher zu installieren. Dieses ist der Descriptor. Er gibt jederzeit Auskunft darüber, wo der Inhalt einer solchen Variablen zu finden ist, bzw. welchen Speicherplatz sie einnimmt. Strings und Felder werden hierbei noch unterschiedlich behandelt.

Der erste Feld-Descriptor enthält in seinen ersten vier Byte die Feldadresse. In den nächsten zwei Byte ist die Anzahl der Dimensionen enthalten. Unter Feldadresse ist hier nicht die Adresse des ersten Byte des ersten Elementes der ersten Dimension zu verstehen, sondern ein weiterer Descriptor, der Informationen über die einzelnen Dimensionen liefert. Die Größe dieses Descriptors ist abhängig von den Dimensionen, die mit DIM vorgesehen wurden.

Am Anfang dieses zweiten Descriptors stehen in jeweils 4 Byte die Dimensionstiefen, rückwärts. Wenn also 3 Dimensionen angegeben wurden, ist der erste Sektor dieses zweiten Descriptors 12 Byte lang. Wurden 4 Dimensionen eingerichtet, ist er 16 Byte lang usw.

Nach der Formel:

```
  Adresse des zweiten Descriptors
+ (Anzahl der Dimensionen * 4)
- (Dimensionsindex * 4)
```

können die einzelnen Elementemengen der mit DIM bestimmten Dimensionen ermittelt werden.

Ab hier unterscheiden sich die Descriptoren für String-Felder von denen für numerische Felder. Für String-Felder schließt sich an diesen ersten Sektor, der in seiner Länge von der Anzahl der Dimensionen abhängig ist, - vorwärts - zusätzlich eine Liste von Descriptoren an, die denselben Aufbau haben, wie der Descriptor für Einzel-Strings. In diesem Falle existieren für jedes Element der ersten Dimension eines String-Arrays ein separater Descriptor. Dessen Lage läßt sich nach der Formel:

```
  Adresse des zweiten Descriptors
+ (Anzahl der Dimensionen * 4)
+ (Elementindex der ersten Dimension * 6)
```

ermitteln. Zu dieser Problematik finden Sie anschließend ein kleines Programm, das Ihnen dabei helfen soll, diese Vorgänge etwas leichter zu verstehen Manchen mag es scheinen, daß bei diesem Programm etwas übertrieben wurde. Dazu gibt es ein Sprichwort:

"Übertreibungen machen deutlich!".

So läßt sich evtl. die Grundstruktur dieses Befehls in seinen Gesetzmäßigkeiten eher verstehen.

```
@Sysfont(1)                        ! Siehe unter ASC
Dim A$(4,3,7)
A$(0,0,0)="A"
A$(1,0,0)="BC"
A$(2,0,0)="DEF"
A$(3,0,0)="GHIJ"
A$(4,0,0)="KLMNO"
D_escripadr%=Arrptr(A$())
A_rrayadr%=Lpeek(D_escripadr%)
Print String$(39,"-")
Print " Feld-Descriptor               : ";D_escripadr%
Print String$(39,"-")
Print " Feldadresse                   : ";Lpeek(D_escripadr%)
Print String$(39,"-")
Print " Anz. d. Dimensionen           : ";Dpeek(D_escripadr%+4)
Print String$(39,"-")
Print " Anz.d.Elem.in 1.Dimension     : ";Lpeek(A_rrayadr%+8)
Print " Anz.d.Elem.in 2.Dimension     : ";Lpeek(A_rrayadr%+4)
Print " Anz.d.Elem.in 3.Dimension     : ";Lpeek(A_rrayadr%)
Print String$(39,"-")
Print " 0/0/0 - Descriptor-Adresse    : ";A_rrayadr%+12
Print " 1/0/0 - Descriptor-Adresse    : ";A_rrayadr%+18
Print " 2/0/0 - Descriptor-Adresse    : ";A_rrayadr%+24
Print " 3/0/0 - Descriptor-Adresse    : ";A_rrayadr%+30
Print " 4/0/0 - Descriptor-Adresse    : ";A_rrayadr%+36
Print String$(39,"-")
Print " 0/0/0 - Adresse mit Varptr    : ";Varptr(A$(0,0,0))
Print " 1/0/0 - Adresse mit Varptr    : ";Varptr(A$(1,0,0))
Print " 2/0/0 - Adresse mit Varptr    : ";Varptr(A$(2,0,0))
Print " 3/0/0 - Adresse mit Varptr    : ";Varptr(A$(3,0,0))
Print " 4/0/0 - Adresse mit Varptr    : ";Varptr(A$(4,0,0))
Print String$(39,"-")
Print " 0/0/0 Adr.im 0/0/0-Descriptor : ";Lpeek(A_rrayadr%+12)
Print " 1/0/0 Adr.im 1/0/0-Descriptor : ";Lpeek(A_rrayadr%+18)
Print " 2/0/0 Adr.im 2/0/0-Descriptor : ";Lpeek(A_rrayadr%+24)
Print " 3/0/0 Adr.im 3/0/0-Descriptor : ";Lpeek(A_rrayadr%+30)
Print " 4/0/0 Adr.im 4/0/0-Descriptor : ";Lpeek(A_rrayadr%+36)
Print String$(39,"-")
Print " String-Länge 0/0/0            : ";Dpeek(A_rrayadr%+16)
Print " String-Länge 1/0/0            : ";Dpeek(A_rrayadr%+22)
Print " String-Länge 2/0/0            : ";Dpeek(A_rrayadr%+28)
Print " String-Länge 3/0/0            : ";Dpeek(A_rrayadr%+34)
Print " String-Länge 4/0/0            : ";Dpeek(A_rrayadr%+40)
Print String$(39,"-")
S_tring.0$=Chr$(Peek(Lpeek(A_rrayadr%+12)))
Print " String 0/0/0                  : ";S_tring.0$
```

```
For I%=0 To 1
 S_tring.1$=S_tring.1$+Chr$(Peek(Lpeek(A_rrayadr%+18)+I%))
Next I%
Print " String 1/0/0                    : ";S_tring.1$
For I%=0 To 2
 S_tring.2$=S_tring.2$+Chr$(Peek(Lpeek(A_rrayadr%+24)+I%))
Next I%
Print " String 2/0/0                    : ";S_tring.2$
For I%=0 To 3
   S_tring.3$=S_tring.3$+Chr$(Peek(Lpeek(A_rrayadr%+30)+I%))
Next I%
Print " String 3/0/0                    : ";S_tring.3$
For I%=0 To 4
   S_tring.4$=S_tring.4$+Chr$(Peek(Lpeek(A_rrayadr%+36)+I%))
Next I%
Print " String 4/0/0                    : ";S_tring.4$
Print String$(39,"-")
Print " 0/0/0 - Backtrailer zeigt auf : ";Lpeek(Lpeek(A_rrayadr%+12)+2)
Print " 1/0/0 - Backtrailer zeigt auf : ";Lpeek(Lpeek(A_rrayadr%+18)+2)
Print " 2/0/0 - Backtrailer zeigt auf : ";Lpeek(Lpeek(A_rrayadr%+24)+4)
Print " 3/0/0 - Backtrailer zeigt auf : ";Lpeek(Lpeek(A_rrayadr%+30)+4)
Print " 4/0/0 - Backtrailer zeigt auf : ";Lpeek(Lpeek(A_rrayadr%+36)+6)
Print String$(39,"-")
L_aenge.3_0%=Len(A$(3,0,0))
If L_aenge.3_0% Mod 2<>0
   Inc L_aenge.3_0%
Endif
Bt.3_0%=Varptr(A$(3,0,0))+L_aenge.3_0%
Print " 3/0/0- Backtrailer zeigt auf  : ";Lpeek(Bt.3_0%)
Print " (mit Len und Varptr ermittelt)"
Print String$(39,"-");Chr$(27);"H"
U=Inp(2)
@Sysfont(2)                  ! Siehe unter ASC
Edit
```

Sie finden in diesem kleinen Listing zwei Sysfont-Aufrufe. Sie bewirken, daß der System-Font zuerst auf die Color-Textgröße 8*8 und zum Ende wieder auf die normale Hires-Größe 8*16 umgestellt wird. Dieses hat, wie sie beim Testen dieses Programms sehen werden, den Vorteil, daß die gesamte Bildschirmausgabe auf eine Screen paßt. Sollten Sie dieses Listing isoliert verwenden, vergessen Sie bitte nicht, diese Prozedur Sysfont mit einzubinden (siehe unter ASC). Es ist übrigens dieselbe Umschaltfunktion, die im Editor-Menü bei Text 16/Text 8 verwendet wird.

Zum zweiten wird in der viertletzten Zeile ein TOS-Befehl verwendet (Chr$(27);"H"), der den Text-Cursor nach Abschluß der letzten PRINT-Zeile auf Home (obere linke Bildschirmecke) setzt, da sonst durch das Zeilen-Scrolling die oberste Textzeile aus dem Bildschirm gedrängt würde. Dieses Unterdrücken des Scrollings ist auch mit einem an den PRINT-Befehl angehängten Semikolon möglich (dazu mehr unter PRINT).

Die enstprechenden Informationen über numerische Variablenfelder sind einschließlich der Dimensionstiefe nach dem gleichen Muster zu ermitteln, wie bei String-Feldern. Nach dem Sektor, der Auskunft über die Tiefen gibt, folgen jedoch bei numerischen Feldern entsprechenden Elemente, bzw. deren Inhalte.

Um nun zu wissen, in welchen Byte-Abständen diese Werte zu finden sind, seien hier noch einmal in Kürze die Variablentypen erwähnt:

```
Realzahl-Variablen          Keine Kennung   => 6 Byte Speicherplatz
                            bzw. in V3.0: # => 8 Byte Speicherplatz
4-Byte-Integer-Variablen    Kennung % => 4 Byte Speicherplatz
Boolesche Variablen         Kennung ! => 2 Byte Speicherplatz
String-Variablen            Kennung $ => 1 Speicher-Byte je Zeichen
```

In V3.0 gibt es zusätzlich noch:

```
1-Byte-Integer-Variablen    Kennung | => 1 Byte Speicherplatz
2-Byte-Integer-Variablen    Kennung & => 2 Byte Speicherplatz
```

24.3 Der Run-Only-Interpreter

Was sicherlich nicht nur die Profis unter den Lesern interessieren wird, ist ein Programm, das Run-Only-Interpreter (engl.: Nur-Lauf-Interpreter) genannt wird. Dies ist ein Interpreter, mit dem man Programme nur laden und starten kann. Die Edition des geladenen Programms ist hiermit nicht möglich. Dieser Run-only-Interpreter ist frei kopierbar und kann also mit den GFA-BASIC-Programmen verbreitet werden. So ist es möglich, diese Programme lauffähig zu machen, ohne den eigentlichen Interpreter besitzen zu müssen.

Auch hier wurde wieder an die Kleinigkeiten gedacht. Es ist nämlich möglich, die mit dem Run-only-Interpreter zusammen auf der Diskette befindlichen BAS-Dateien durch einfaches Anklicken zu starten. Auf diese Weise unterscheiden sich GFA-BASIC-Programme in der Bedienung für den Anwender überhaupt nicht mehr von compilierten oder assemblierten Programmen. Man klickt sie an und sie starten selbsttätig. Was man bei anderen BASIC-Dialekten meist schmerzlich vermissen muß, ist hier ein Kinderspiel: der Auto-Start (er funktioniert übrigens auch beim Editor-Interpreter).

24.4 Der Direktmodus

Hiermit steht Ihnen Eingabemöglichkeit zur Verfügung, die nicht auf den Befehl RUN wartet, sondern alle Eingaben sofort (direkt) nach deren Formulierung und anschließendem Betätigen der <Return>-Taste ausführt. Es ist hier nicht möglich, Befehlsketten zu bilden, die dann der Reihe nach abgearbeitet werden. Es sei denn, man hat vorher im Programmeditor eine PROCEDURE definiert, die dann vom Direktmodus aus mit GOSUB angesprungen werden kann.

FOR..NEXT-Schleifen z.B. sind also im Direktmodus nicht machbar, da sie mindestens zwei getrennte Befehle benötigen. Welche Befehle und Anweisungen hier nicht möglich sind, wird Ihnen der Interpreter anchließend an die Fehleingabe in einer Alert-Box melden.

Das mag alles auf den ersten Blick etwas umständlich wirken, hat jedoch den unübersehbaren Vorteil, daß es dadurch möglich ist, sich eine eigene Bibliothek an Funktionen und Hilfsprogrammen zusammenzustellen, die dann einfach über einen GOSUB-Sprungbefehl angesprochen werden können.

Ein Mangel ist, daß unter ungünstigsten Umständen ein Systemfehler, der zwar gemeldet wurde, aber nicht komplett abgefangen werden konnte, dazu führen kann, daß der nicht mehr zu verhindernde Absturz erst bei der Umschaltung zum Direktmodus stattfindet.

Zur allgemeinen Beruhigung: dieser Umstand tritt äußerst selten auf, da dieser Interpreter über ein "fast" perfektes Error-Handling verfügt.

Des weiteren ist im Farbmodus zu beachten, daß die Schriftfarben für den Editor und den Direktmodus aus verschiedenen Farbregistern bezogen werden. Wenn also über das Register 1 die Ausgabetextfarbe so eingestellt wurde, daß sie mit dem Hintergrund (Register 0) identisch ist, kann es sein, daß Sie den Editortext (Register 3) noch lesen können und im Direktmodus keine Schrift mehr zu erkennen ist. Dieser Schaden kann realtiv leicht behoben werden, indem man nun "blind" SETCOLOR 1, Farbe mit einer Farbe eingibt, die sich vom Hintergrund abhebt. Derselbe Schritt ist in V2.xx vorzunehmen, wenn sich im Editor kein Text mehr erkennen läßt. In V3.0 ist dies unnötig, da der Interpreter bei Rückkehr zum Editor wieder die Ausgangsfarben restauriert.

Wollen Sie aus dem Direktmodus wieder zum Editor wechseln, geben Sie einfach Ed+<Return> ein oder drücken <Esc> und anschließend

<Return>. Außerdem erreichen Sie den Editor auch durch die Break-Funktion <Control><Shift><Alternate>.

Wie Sie im Laufe ihrer wachsenden Routine an diesem Interpreter erkennen werden, kann man angesichts der erheblichen Vorteile dieses Modus die Nachteile fast vergessen.

Die jeweils letzte Eingabezeile im Direktmodus kann durch <Undo> wieder aufgerufen werden.

In Version V3.0 ist ein sogenannter History-Modus eingebaut, der es erlaubt, sich durch Betätigung der <Pfeil-hoch>- bzw. <Pfeil-runter>-Cursor-Taste die letzten acht Direkt-Befehle anzeigen zu lassen. Die angezeigte Zeile kann neu editiert werden und/oder durch <Return> erneut gestartet werden. Außerdem kann nun durch <Insert> zwischen Einfüge- und Überschreibmodus gewählt werden.

Wie auch bei KEYDEF habe ich mir hier erlaubt, drei kleine Editor-Erweiterungen für die V3.0-Version zu produzieren. Beachten Sie dazu auch die Anmerkungen unter KEYDEF.

```
PROCEDURE dsave(datname$)
  ' Speichert den kompletten Zeilen-Puffer (8 Zeilen) des
  ' History-Modus auf der Diskette ab.
  ' Datname$ =  Name der History-Datei.
  '             Der Name (Pfad) darf keine Extension tragen,
  '             da diese automatisch auf .HIS gesetzt wird.
  '
  IF RIGHT$(datname$,4)<>".HIS"
    IF LEN(datname$)<9 AND INSTR(datname$,".")=0
      BSAVE datname$+".HIS",BASEPAGE+124474,256*8
    ENDIF
  ENDIF
RETURN
PROCEDURE dload(datname$)
  ' Lädt einen kompletten, mit Dsave gespeicherten History-Block
  ' von der Diskette. Die geladenen Zeilen können anschließend
  ' durch die beiden Vertikal-<Pfeiltasten> gewählt werden.
  ' Datname$ =  Name der zu ladenden History-Datei.
  '             Der Name (Pfad) darf keine Extension tragen,
  '             da diese automatisch auf .HIS gesetzt wird.
  '
  IF EXIST(datname$+".HIS")
    BLOAD datname$+".HIS",BASEPAGE+124474
  ENDIF
RETURN
PROCEDURE dline(idx%)
  ' Kleines Bonbon! Diese Routine belegt die History-Zeile
  ' mit dem Index idx% (1 - 8) mit dem zuletzt durch
  ' <Control><P> (siehe 24.5 "Der Editor") gelöschten
  ' String(-Teil).
  ' Fkey% = Index der gewünschten History-Zeile (1 - 8)
```

```
  '
  LOCAL bf$,bp%
  IF idx%=>0 AND idx%<=8
    bp%=BASEPAGE
    bf$=SPACE$(LEN(CHAR{{bp%+24}}))
    BMOVE {bp%+24},bp%+124474+(MAX(1,idx%)-1)*256,LEN(bf$)
    POKE bp%+124474+(MAX(1,idx%)-1)*256+LEN(bf$),0
  ENDIF
RETURN
```

24.5 Der Editor

Neben einer gelungenen und programmierfreundlichen Syntax spielt der Programm-Editor die zweitgrößte Rolle bei einer Computersprache. Was soll ich mit einer noch so genialen Syntax, wenn die Funktionsweise des Editors mir Kabinettstückchen abverlangt, um meine Programmvorstellungen Realität werden zu lassen. In diesem Punkt gibt es wenig am GFA-BASIC auszusetzen.

Die V3.0-Version des GFA-BASIC weist ein Fülle an Veränderungen auf. So auch der Editor. Um innerhalb der Beschreibung seiner Funktionen nicht das völlige Chaos zu veranstalten, hielt ich es für angebracht, dieses Kapitel zu unterteilen. Sie finden im ersten Teil alle Funktionen, die sowohl für die V2.xx-Versionen, als auch für die V3.0-Version unverändert - bis auf kleine Änderungen an der Replace- und der Cursor-TAB-Funktion - gültig sind.

Im zweiten Teil sind dann alle Änderungen und zusätzlichen Funktionen der V3.0-Version aufgeführt.

24.5.1 Editor-Optionen in V2.xx und V3.0

```
| Save    | Save,A  | Quit    | New     |Blk Sta |Replace | Pg up   |Text 8  | Direct | Run  |
| Load    | Merge   | Llist   | Block   |Blk End | Find   | Pg down |Insert  | Flip   | Test |
Xt%=2-Sgn(Xbios(4))
Yt%=Min(2,3-Xbios(4))
Deftext ,0,,12/(Yt%+1)
Deffill ,2,4
Text 12/Xt%,14/Yt%,350/Xt%,"'ROUND' - elliptische Verformung"
Pellipse 320/Xt%,200/Yt%,250/Xt%,90/Yt%
Graphmode 1
@Round(12/Xt%,0,365/Xt%,16/Yt%,320/Xt%,200/Yt%,260/Xt%,100/Yt%,130,410,2)
Procedure Round(Xl%,Yo%,Xr%,Yu%,Xp%,Yp%,Xrd%,Yrd%,Sw%,Ew%,Gmd%)
  ' Xl% = linke X-Koordinate des Ausschnitts
  ' Yo% = obere Y-Koordinate des Ausschnitts
  ' Xr% = rechte X-Koordinate des Ausschnitts
  ' Yu% = untere Y-Koordinate des Ausschnitts
  ' Xp% = Mittelpunkt-X-Koordinate der Ellipse
  ' Yp% = Mittelpunkt-Y-Koordinate der Ellipse
  ' Xrd%= horizontaler Ellipsenradius
  ' Yrd%= vertikaler Ellipsenradius
  ' Sw% = Startwinkel in Grad (0 = rechtes Ende der Horizontalachse)
  ' Ew% = Endwinkel in Grad (360 = rechtes Ende der Horizontalachse)
  ' Gmd%= Ausführungsmodus 1=REPLACE / 2=TRANSPARENT
  Local I,J%,K,H,Stp,Stp1
  If Sw%=Ew%
    Ew%=Sw%+359
  Endif
  Stp=-Min(1,(Ew%-Sw%+1)/(Xr%-Xl%))
  Stp1=Min(1,1/((Ew%-Sw%+1)/(Xr%-Xl%)))
  Stp2=(Max(Xrd%+J%,Yrd%+J%)/30+1.0E-20)
  For I=90-Sw% To 90-Ew% Step Stp
    If I=<90-Ew%-(Stp*Stp2)
      Sub Stp2,Stp
    Endif
    For J%=0 To Abs(Yu%-Yo%)
      R.pnt%=Point(Xl%+H,Yo%+(Yu%-Yo%)-J%)
      If (R.pnt%>0 And Gmd%=2) Or Gmd%=1
        Color R.pnt%
        For K=0 To 1 Step 1/Stp2
          Plot Xp%+Sin((I+K)*Pi/180)*(Xrd%+J%),Yp%+Cos((I+K)*Pi/180)*(Yrd%+J%)
        Next K
      Endif
    Next J%
    Add H,Stp1
  Next I
Return
```

Cursor rechts

==> <Rechtspfeil>

Cursor links

==> <Linkspfeil>

Cursor an Zeilenanfang

==> <Control> + <Linkspfeil>

Cursor an Zeilenende

==> <Control> + <Rechtspfeil>

Ist die Programmzeile länger als 79 Zeichen, wird die Zeile während der Edition jeweils um ein Zeichen nach links versetzt. Nach Abschluß der Zeile erscheint am rechten Bildrand in dieser Zeile ein Pfeil, der so andeutet, daß die Zeilenlänge mehr als 79 Zeichen beträgt.

Cursor-rechts-Tabulator
==> <Tab>

In V3.0 ist es nun möglich, hiermit den Cursor hinter dem Zeilenende zu positionieren.

Cursor-links-Tabulator
==> <Control> + <Tab>

Cursor eine Zeile aufwärts
+ Syntax-Check
==> <Pfeil-hoch>

Cursor eine Zeile abwärts
+ Syntax-Check
==> <Pfeil-runter> oder <Return>

Cursor beliebig positionieren
+ Syntax-Check bei Zeilenwechsel
==> Mausklick auf gewünschte Position

Wenn der Cursor nicht an den Zeilenanfang bewegt werden soll, sondern direkt an die Mauszeigerposition, ist es ratsam, die Maustaste kurze Zeit gedrückt zu halten, da die Sprungorientierung sich zuerst auf den Zeilenanfang bezieht und erst in zweiter Instanz auf die Mauszeigerposition innerhalb der Zeile.

Cursor an Seitenanfang
+ Syntax- und Struktur-Check
==> <ClrHome>

Cursor eine Seite rückwärts
+ Syntax- und Struktur-Check

==> <Control> + <Pfeil-hoch> oder <Shift> + <F7>
oder "Pg up" im Editor-Menü anklicken.

Cursor eine Seite vorwärts
+ Syntax- und Struktur-Check
==> <Control> + <Pfeil-runter> oder <F7>
oder "Pg down" im Editor-Menü anklicken.

Cursor an Programmanfang
+ Syntax- und Struktur-Check
==> <Control> + <ClrHome>

Cursor an Programmende
+ Syntax- und Struktur-Check
==> <Control> + <Z>

Programm speichern
==> <Shift> + <F1> oder "Save" im Editor-Menü anklicken.

V2.xx: Programm-Code wird mit Extension .BAS gespeichert.
V3.00: Programm-Code wird mit Extension .GFA gespeichert.

Das Programm wird als einheitlicher Speicherblock auf die Diskette geschrieben, ohne die Interpreter-Kontrolle zu durchlaufen. Dadurch ergibt sich eine erheblich kürzere Speicherzeit als bei Save,A.

Programm laden
==> <F1> oder "Load" im Editor-Menü anklicken.

V2.xx: Programm-Code mit Extension .BAS laden.
V3.00: Programm-Code mit Extension .GFA laden.

Hier wird ebenfalls die Interpreter-Kontrolle übersprungen. Ein hiermit geladenes Programm überschreibt das aktuelle Programm. Ggfs.

sollte vorher das aktuelle Programm gesichert werden (vgl. LOAD). Der Lade-Vorgang läßt sich, nachdem er in Gang gebracht ist, nur noch durch einen Reset stoppen.

Textdatei speichern

==> <Shift> + <F2> oder Save,A im Editor-Menü anklicken.

Programm in ASCII mit Extension .LST speichern.

Textdatei laden

==> <F2> oder Merge im Editor-Menü anklicken.

Programm in ASCII mit Extension .LST laden.

Ein hiermit geladenes Programm wird in das aktuelle Programm ab Cursor-Position eingefügt. Gemergte Zeilen, die nicht korrekt interpretiert werden können, werden mit dem Symbol '==>' markiert. Zeilen mit darin enthaltenen Null-Bytes oder zu großer Länge (>255) werden abgewiesen und der Merge-Vorgang abgebrochen.

Interpreter-Arbeit beenden

==> <Shift> + <F3> oder "Quit" im Editor-Menü anklicken.

Programm-Listing ausdrucken

==> <F3> oder "Llist" im Editor-Menü anklicken.

Vorsicht: Ist der Drucker nicht angeschlossen oder nicht On Line, so kehrt das System erst nach ca. einer halben Minute zum Interpreter zurück. Also, nicht gleich den RESET-Knopf drücken! Antwort NEIN kehrt zum Interpreter zurück.

Programm löschen

==> <Shift> + <F4> oder "New" im Editor-Menü anklicken.

Blockoperationen (Blockmenü öffnen)

==> <F4> oder "Block" im Editor-Menü anklicken.

Wurde mit Blk Sta und Blk End ein Textblock definiert, so erscheint das Block-Menü. Fehlt eine der beiden Definitionen, so erscheint

"Block ???". Wird bei offenem Blockmenü eine ungültige <Taste> gedrückt oder außerhalb des Menüs in das Editor-Fenster geklickt, so verschwindet es wieder.

Block an Cursor-Position kopieren
==> <C> oder "Copy" im Block-Menü anklicken.

Block an Cursor-Position bewegen
==> <M> oder "Move" im Block-Menü anklicken.

Block als ASCII-Text speichern
==> <W> oder "Write" im Block-Menü anklicken.

Block ausdrucken
==> <L> oder "Llist" im Block-Menü anklicken.

Block-Anfang finden
==> <S> oder "Start" im Block-Menü anklicken.

Block-Ende finden
==> <E> oder "End" im Block-Menü anklicken.

Block löschen
==> <Control> + <D> oder "Delete" im Block-Menü anklicken.

Block-Anfang definieren
==> <Shift> + <F5> oder "Blk Sta" im Editor-Menü anklicken.

Block-Ende definieren
==> <F5> oder "Blk End" Im Editor-Menü anklicken.

Sollte sich bei Aufruf der beiden vorangegangenen Blockoperationen der Cursor innerhalb des Blocks befinden, wird die Fehlermeldung "Cursor in Block" ausgegeben. Wird das Block-Ende über oder in der Block-Anfangszeile definiert, wird die bisherige Block-Definitionen rückgängig gemacht.

In V3.0 wird der Block im Farbmodus nicht mehr gerastert, sondern farbig dargestellt.

Ausdruck ab Cursor-Position finden und ersetzen
==> <Shift> + <F6> oder "Replace" im Editor-Menü anklicken.

Im Anschluß an die String-Eingabe kann dann durch <Control> + <F> die Suche ausgelöst werden. In V2.xx: <Control> + <R> ersetzt ggfs. den gefundenen Ausdruck durch den Ersatzausdruck.

In V3.0: Bei Eingabe des Such- und des Ersatz-Strings erscheint der letzte String als Vorgabe erneut in der Eingabezeile. Die Eingabezeile kann durch <Esc> gelöscht werden. <Control> + <E> ersetzt ggfs. den gefundenen Ausdruck durch den Ersatzausdruck. Strings innerhalb "versteckter" Prozeduren werden nicht gefunden oder ersetzt. Wird der Ausdruck gefunden, wird die entsprechende Programmzeile angezeigt und der Cursor befindet sich auf dem ersten Zeichen des Ausdrucks. Ist der Ausdruck im weiteren Listing nicht mehr enthalten, bleibt der Cursor an der Suchstart-Position stehen.

Der Find- oder Replace-String kann eine Länge von 60 Zeichen nicht überschreiten. Die Edition des Find- oder Replace-Strings erfolgt wie üblich durch die Tasten <Delete>, <Backspace> und die beiden Horizontal-Pfeiltasten. Zudem kann der Cursor mit den Vertikal-Pfeiltasten jeweils an den Anfang oder das Ende des Strings bewegt werden. Soll die String-Eingabe abgebrochen werden, löschen Sie evtl. schon eingegebene Zeichen (<Delete> oder <Backspace>) und quittieren Sie mit <Return> oder drücken Sie die <Esc>-Taste und anschließend <Return>. Um das gesamte Programm zu durchsuchen, muß ggfs. der Cursor durch gleichzeitiges Drücken von <Control> und <ClrHome> am Programmanfang positioniert werden.

Ausdruck ab Cursor-Position finden
==> <F6> oder "Find" im Editor-Menü anklicken.
<Control> + <F> sucht jeweils nächsten Ausdruck.

Zu den folgenden drei V3.0-Prozeduren beachten Sie die Anmerkungen und Prozedur-Beschreibungen unter KEYDEF bzw. in der Beschreibung des V3.0-Direktmodus.

```
PROCEDURE fsave(datname$)
  ' Speichert den aktuellen Find- und Replace-String
  ' in einer beliebigen Disk-Datei ab.
```

```
    ' Datname$ = Beliebiger Dateiname ohne Extension.
    '            Die Extension .FND wird automatisch gesetzt.
  IF RIGHT$(datname$,4)<>".FND"
  IF LEN(datname$)<9 AND INSTR(datname$,".")=0
     BSAVE datname$+".FND",BASEPAGE+93364,128
    ENDIF
    ENDIF
  RETURN
  PROCEDURE fload(datname$)
    ' Lädt den durch Fsave gespeicherten Find-
    ' und Replace-String von der Diskette und
    ' trägt die beiden Strings in dem internen
    ' String-Puffer ein.
    ' Datname$ = Beliebiger Dateiname ohne Extension.
    '            Extension .FND wird automatisch gesetzt.
    '
    IF EXIST(datname$+".FND")
      BLOAD datname$+".FND",BASEPAGE+93364
    ENDIF
  RETURN

  PROCEDURE fline(idx%)
    ' Diese Routine belegt entweder den internen
    ' Find-String-Puffer (idx% = 0) oder den internen
    ' Replace-String-Puffer (idx% = 1) mit max. 61
    ' Zeichen des zuletzt durch <Control><P>
    ' (siehe 24.5 "Der Editor") gelöschten String(-Teils).
    ' Idx%:
    '           0 = Ziel ist Find-String
    '           1 = Ziel ist Replace-String
    '
    LOCAL bf$,bp%
    IF idx%=>0 AND idx%<=1
      bp%=BASEPAGE
      bf$=SPACE$(LEN(CHAR{{bp%+24}}))
      BMOVE {bp%+24},bp%+93364+idx%*64,MAX(61,LEN(bf$))
      POKE bp%+93364+idx%*64+MAX(61,LEN(bf$))+1,0
    ENDIF
  RETURN
```

Editor-Zeilenhöhe ändern (nur in Hires)

==> <Shift> + <F8> oder "Text 16" bzw.
"Text 8" im Editor-Menü anklicken (in V3.0 : "Txt 16").

Text-Eingabemodus ändern

==> <F8> oder "Insert" bzw.
"Overwrt" im Editor-Menü anklicken.

Die eingegebenen Zeichen können verschiedene Auswirkungen auf den schon bestehenden Programmtext haben. Im Insert-Modus (insert; engl.: einfügen) wird das einzugebende Zeichen an der Cursor-Position in den Text eingefügt. Der rechts davon liegende Zeilenrest wird um

eine Stelle nach rechts verschoben. Im Overwrite-Modus (overwrite; engl.: überschreiben) wird das Zeichen unter dem Cursor durch das neue Zeichen überschrieben.

Direktmodus einschalten

==> <Shift> + <F9> oder <Esc>
oder "Direct" im Editor-Menü anklicken.

Ausgabe-Fenster zeigen

==> <F9> oder "Flip" im Editor-Menü anklicken.

Das zweite Interpreter-Fenster, das zur Ausgabe von Grafik- oder Textoperationen dient, wird gezeigt. Dieser Vorgang bleibt ohne weitere Auswirkung auf die Editor-Arbeit. Nach Drücken einer beliebigen Taste oder einem MausKlick unterhalb des Editor-Menüs kehrt der Interpreter zum Editor zurück.

Programm starten

==> <Shift> + <F10> oder "Run" im Editor-Menü anklicken.

Aktuelles Programm nach Syntax-Check starten.

Schleifenstruktur testen

==> <F10> oder "Test" im Editor-Menü anklicken.

Syntax- und Struktur-Check ohne Start.

Letzte Änderung löschen + Struktur ordnen
+ Syntax- und Struktur-Check

==> <Undo>

Wurde eine Programmzeile nach Änderung noch nicht mit <Return> oder einer Vertikal-Pfeiltaste verlassen, kann diese Änderung wieder ungültig gemacht werden. Gleichzeitig wird das Programm komplett strukturiert. Wurden Programmteile verändert, ersetzt oder gelöscht, kann es sein, daß die sichtbare Programmstruktur nicht mehr korrekt geordnet ist. Mit dieser Funktion wird das Programm wieder in Ordnung gebracht.

Leerzeile einfügen

==> <Insert>

Hierdurch entsteht unter dem Cursor eine Leerzeile. Diese Zeile wird dann wie jede andere neue Zeile behandelt. Nach Texteingabe und <Return> als Zeilenabschluß wird solange eine neue Leerzeile eingefügt bis diese dann leer mit <Return>, einer der Vertikal-Pfeiltasten oder <Undo> geschlossen wird.

Aktuelle Cursor-Zeile (V2.xx: unwiderruflich) löschen

==> <Control> + <Delete>

Zeichen links vom Cursor löschen

==> <Backspace>

Zeichen unter dem Cursor löschen

==> <Delete>

24.5.2 Editor-Erweiterungen in V3.0

```
   Save   Save,A  Quit   New    BlkSta Replac  Pg ↑   Text 8 Direct  Run    00:28:35
   Load   Merge   Llist  Block  BlkEnd Find    Pg ↓   Insert Flip    Test          3
xt%=2-SGN(XBIOS(4))
yt%=MIN(2,3-XBIOS(4))
DEFFILL ,2,2
PBOX 0,0,640/xt%,400/yt%
DEFFILL ,2,1
@chart(110/xt%,160/yt%,100/xt%,-46/yt%,10/yt%,2,1,2,5)
@chart(110/xt%,226/yt%,70/xt%,-30/yt%,60/yt%,2,1,2,5)
@chart(110/xt%,280/yt%,40/xt%,-14/yt%,50/yt%,2,1,2,5)
FOR j%=150 TO 280 STEP 20
  ADD k%,12
  FOR i%=200 TO 500 STEP 40
    @chart((i%+k%)/xt%,j%/yt%,10/xt%,4/yt%,(RANDOM(60)+10)/yt%,2,4,2,1)
  NEXT i%
NEXT j%
PROCEDURE chart(c.x%,c.y%,c.rx%,c.ry%,c.h%,c.fs1%,c.fm1%,c.fs2%,c.fm2%)
  ' Säulengrafik
  ' C.x%  = Fuß-Mittelpunkt-X-Koordinate
  ' C.y%  = Fuß-Mittelpunkt-Y-Koordinate
  ' C.rx% = Horizontal-Radius
  ' C.ry% = Vertikal-Radius (auch Minuswerte möglich)
  ' C.h%  = Säulenhöhe
  ' C.fs1%= Füllstil der Säule
  ' C.fm1%= Füllmuster der Säule
  ' C.fs2%= Füllstil des Deckels
  ' C.fm2%= Füllmuster des Deckels
  LOCAL i,j%
  DIM c.px%(38),c.py%(38)
  DEFFILL ,c.fs1%,c.fm1%
  FOR i=-90 TO 90 STEP 10
    INC j%
    c.px%(j%)=c.x%+SIN(i*PI/180)*c.rx%
    c.py%(j%)=c.y%+COS(i*PI/180)*c.ry%
    c.px%(19+j%)=c.x%-SIN(i*PI/180)*c.rx%
    c.py%(19+j%)=c.y%-c.h%+COS(i*PI/180)*c.ry%
  NEXT i
  c.px%(0)=c.px%(38)
  c.py%(0)=c.py%(38)
  POLYFILL 38,c.px%(),c.py%()
  DEFFILL ,c.fs2%,c.fm2%
  IF c.ry%>0
    PELLIPSE c.x%,c.y%-c.h%,c.rx%,c.ry%
  ELSE
    PELLIPSE c.x%,c.y%-1,c.rx%,c.ry%
  ENDIF
  ERASE c.px%()
  ERASE c.py%()
RETURN
```

Die Menüleiste

Die Editor-Menüleiste der V3.0-Version weist vier wesentliche Veränderungen auf.

Menüpunkt ganz links oben: Hier befindet sich das Atari-Symbol. Wird es durch Mausklick angewählt, gelangt man in ein Pull-Down-Menü mit zwei Menü-Titeln.

Titel 1 (ebenfalls Atari-Symbol)

Menüpunkt 1: "GFA-BASIC 3.00 D"
Es erscheint eine Alert-Box mit der Auswahl "Editor" und "Menü". Durch "Editor" gelangt man in den Programmeditor zurück, durch "Menü" wieder ins Pull-Down-Menü. Menüpunkte 2 bis 7 (falls vorhanden): Evtl. geladene Accessories.

Titel 2 (GFA-BASIC)

Menüpunkt 1: "Save"
Es wird in den Editor gesprungen und eine Fileselect-Box zum Abspeichern des aktuellen Programms aufgerufen.

Menüpunkt 2: "Load"
Es wird in den Editor gesprungen und eine Fileselect-Box zum Laden eines GFA-BASIC-Programms aufgerufen.

Menüpunkt 3: "Deflist"
Es erscheint eine Alert-Box, durch welche die gewünschte DEFLIST-Einstellung initialisiert werden kann (vgl. DEFLIST).

Menüpunkt 4: "Neue Namen"
Es erscheint eine Alert-Box, durch welche bestimmt werden kann, ob bei Einführung neuer Variablen-, Label-, Prozedur- oder Funktionsnamen eine Meldung ausgegeben werden soll. Anhand dieser Meldung kann dann entschieden werden, ob der neue Name als Bestandteil in das Programm übernommen wird oder nicht. Falls nicht, wird die Eingabe als Syntax-Error abgewiesen. Bei Editor-Start ist diese Funktion ausgeschaltet.

Menüpunkt 5: "Editor"
Wird dieser Punkt gewählt, springt der Interpreter in den Editor.

Menüpunkt ganz links darunter
Dieser zwei Zeichen breite Menüpunkt ist bei GFA-Start üblicherweise leer. In der linken Position erscheint bei eingeschaltetem Caps-Lock ein Aufwärtspfeil und in der rechten bei eingeschaltetem Num-Lock ein 'Hoch'-Zeichen (^). Diese beiden Pfeile (Modi) können durch Anklicken an- und ausgeschaltet werden.

Menüpunkt ganz rechts oben
In diesem acht Zeichen langen Feld erscheint die aktuelle Uhrzeit. Sie kann verändert werden, indem man das Feld anklickt. Der Cursor steht dann auf dem ersten Feld der Stunden-Angabe und es wird die Uhrzeit-Eingabe erwartet. Nach <Return> wird die eingegeben Uhrzeit übernommen. Wird statt dessen die Eingabe durch <Esc> abgebrochen, wird die alte Zeitangabe wieder restauriert.

Menüpunkt ganz rechts darunter
In diesem Feld erscheint die Nummer der Programmzeile, auf welcher sich der Cursor momentan befindet. Es ist möglich, eine beliebige Nummer einzugeben, indem man das Feld anklickt oder <Control> und <G> drückt. Es wird dann die Eingabe einer Zeilennummer erwartet. Nach <Return> wird die Zeile - sofern vorhanden - angesprungen.

V3.0-Programm speichern/laden

Die Codierung der SAVE-Programme in Version V3.0 wurde geändert. V2.xx-Programme mit der Extension .BAS lassen sich durch den V3.0-Editor nicht mehr laden. Auch wenn es bei größeren Programmen sehr umständlich ist, es bleibt nichts anderes übrig, als die .BAS-Programme von der V2.xx-Version mit Save,A als ASCII-Datei abspeichern zu lassen und diese anschließend mit Merge in den V3.0-Editor zu laden. Mit Save (oder dem SAVE-Befehl) abgespeicherte V3.0-Programme erhalten - sofern keine andere vorgegeben wurde - die Extension .GFA.

Druckersteuerung

Die Editor-Funktion Llist (<F3>) wurde erweitert. Es ist möglich, innerhalb des Listings verschiedene Formatvorgaben zur Druckerausgabe des Programmlistings zu machen. Es handelt sich um sogenannte Punkt-Befehle, die wie normale Programmzeilen in das Listing eingefügt werden können. Wird am Zeilenanfang ein Punkt ohne sinnvolle Druckersteuerung verwendet (z.B.: ...ABC), wird die Zeile vom Interpreter ignoriert. x steht im folgenden als Platzhalter für eine Ziffer:

```
.ll xx      - Maximale Zeilenlänge = xx Zeichen
.pl xx      - Maximale Seitenlänge = xx Zeilen
.ff xxx     - Ersatz-ASCII für FF (FormFeed)
```

Erwartet der Drucker einen anderen Steuer-Code für Form Feed als 12, kann der benötigte Wert übergeben werden (Default = .ff 012).

```
.he text$  - Kopf-(head-)zeilentext in text$
.fo text$  - Fuß-(foot-)zeilentext in text$
```

In Kopf- und Fußzeilentext können Platzhalter eingefügt werden:

```
\xxx        - Textzeichen mit dem ASCII xxx
\d          - Aktuelles Datum
\t          - Aktuelle Uhrzeit
#           - Aktuelle Seitennummer
```

Sollen die hier verwendeten Sonderzeichen \ und # ebenfalls ausgegeben werden, ist ihnen ein Backslash \ voranzustellen (\\ bzw. \#). Der vorangestellte Backslash wird dann nicht gedruckt.

```
.lr xx      Links freien Rand lassen = xx Zeichen
.l-         Folgende Zeilen nicht mit ausdrucken
.l+         .l- wieder aufheben. Ab hier wieder drucken
.nx         Zeilennummerierung einschalten
```

x steht hier für einen Wert von 1 bis 9, der die maximale Stellenanzahl der Zeilennummern angibt

```
.n0         .nx wieder aufheben. Zeilennummerierung abschalten
```

Tastatur-Kommandos

Procedure "Verstecken" (Folding)
==> Cursor auf Prozedurkopfzeile, dann <Help> drücken.

Kopfzeile wird durch > gekennzeichnet. Diese Kennzeichnung bleibt auch beim Speichern durch Save oder Save,A im Programm-File er-

halten und wird beim Laden durch Load oder Merge erkannt, so daß danach wieder nur die Kopfzeile erscheint. Wird zusätzlich zur <Help>-Taste die <Control>-Taste gedrückt, so werden **alle** Prozeduren ab der aktuellen Zeile "zusammengefaltet".

Procedure "Listen" (Unfolding)

==> Cursor auf Prozedurkopfzeile, <Help> drücken oder
> am Zeilenanfang löschen oder ganze Kopfzeile löschen.

Wird zusätzlich zur <Help>-Taste die <Control>-Taste gedrückt, so werden **alle** Prozeduren ab der aktuellen Zeile "auseinandergefaltet".

NumLock-Modus an/aus

==> <Contrl> + <−>, dann <Contrl> + <(>

Ist der NumLock-Modus aktiv (erkennbar am ^-Zeichen in der Editor-Menüleiste), können die folgenden NumLock-Kommandos ohne zusätzlichen Druck auf die <Contrl>-Taste ausgelöst werden.

NumLock-Kommandos

(Mit den angegebenen Ziffern ist die jeweilige Taste auf dem Ziffertastenblock gemeint!)

Cursor nach links

==> <Control> + <4>

Cursor nach rechts

==> <Control> + <6>

Cursor eine Zeile aufwärts

==> <Control> + <8>

Cursor eine Zeile abwärts

==> <Control> + <2>

Cursor an Programmanfang

==> <Control> + <7>

Cursor an Programmende
==> <Control> + <1>

Cursor eine Seite rückwärts
==> <Control> + <9>

Cursor eine Seite vorwärts
==> <Control> + <3>

Leerzeichen einfügen
(entspricht <Insert>)

==> <Control> + <0>

Cursor-Zeichen löschen (entspricht <Delete>)
==> <Control> + <.>

Neue Control-Sequenzen

Aktuelle Cursor-Zeile löschen (entspricht <Control>+<Delete>)
==> <Control> + <Y>

Zeile restaurieren
==> <Control> + <U>

Die zuletzt mit <Control> + <Delete> oder <Control> + <Y> gelöschte Zeile wird an der aktuellen Cursor-Position wieder ausgegeben. Diese Operation ist auch mehrfach mit derselben Zeile möglich.

Zeilenrest löschen
==> <Control> + <P>

Der Zeilenrest ab der aktuellen Cursor-Position (inkl. Cursor-Position) wird gelöscht und in einem internen Puffer gespeichert.

Sofern die Zeile sofort nach Löschung ohne weitere Änderungen verlassen wird, bleibt die Zeile im vorherigen Zustand erhalten. Die vor der Löschung komplette Zeile wird wieder restauriert, sobald man den Cursor wieder in die Zeile zurücksetzt. Vorsicht: wenn Sie nicht in die Zeile zurückgehen, ist sie trotzdem noch vollständig im Programmspeicher und wird dann ggfs. später (evtl. beim Blättern) wieder komplett angezeigt. Soll die Zeile in der nach Löschung sichtbaren Form in den Speicher übernommen werden, so muß nach Löschung mindestens 1 Leerzeichen am sichtbaren Zeilenende eingegeben werden (funktioniert nicht immer?).

Zeilenrest einfügen
==> <Control> + <O>

Der durch <Control><P> gelöschte Zeilenrest bleibt solange im Puffer erhalten, bis ein neuer Teil durch <Control><P> gelöscht wird.

Diese zuletzt gelöschte Teilzeile kann jederzeit durch <Control><O> an der aktuellen Cursor-Position ausgegeben werden (funktioniert nicht im Direktmodus).

Leerzeile einfügen (entspricht <Insert>)
==> <Control> + <N>

Blockoperationen, Blockmenü öffnen (entspricht <F4>)
==> <Control> + <Q>

Block-Anfang definieren (entspricht <Shift>+<F5>)
==> <Control> + <B>

Block-Ende definieren (entspricht <F5>)
==> <Control> + <K>

Cursor eine Seite rückwärts (entspricht <Control>+<Pfeil-hoch>, bzw. <Shift> + <F7>)
==> <Control> + <R>

Cursor eine Seite vorwärts (entspricht <Control>+<Pfeil-runter>, bzw. <F7>)
==> <Control> + <C>

Ausdruck ab Cursor-Position finden (entspricht <F6>)
==> <Shift> + <Control> + <F>

Im Anschluß an die String-Eingabe sucht <Control> + <F> den jeweils nächsten Ausdruck.

Ausdruck ab Cursor-Position finden und Ersetzen (entspricht <Shift>+<F6>)
==> <Shift> + <Control> + <E>

Im Anschluß an die String-Eingabe kann dann durch <Control> + <F> die Suche ausgelöst werden. <Control> + <E> ersetzt ggfs. den gefundenen Ausdruck durch den Ersatzausdruck.

Zeilensprung
==> <Control> + <G>

Öffnet das Feld der Zeilennummern-Anzeige rechts in der Editor-Menüzeile. Durch Eingabe einer Zeilennummer mit anschließendem <Return> wird - sofern vorhanden - zu der angegebenen Zeile gesprungen.

Editor-Zeilenmarkierung setzen
==> <Control> + <1> (Hauptzifferntaste)
bis <Control> + <6> (Hauptzifferntaste)

Es können durch <Control> und die <Ziffer>n 1 bis 6 aus der Hauptziffernleiste bis zu sechs - unsichtbare - Marken im Editor auf beliebige Programmzeilen (jeweils die aktuelle Cursor-Zeile) gesetzt werden (siehe unten <Alternate>+<Ziffer>).

Alternate-Sequenzen

Editor-Zeilenmarkierung anspringen:

==> <Alternate> + <1>
bis <Alternate> + <6>

Es kann die durch die jeweilige Ziffer (aus der Hauptziffernleiste) markierte Zeile angesprungen werden, falls die hier angegebene Ziffer vorher durch <Control>+<Ziffer> (s.o.) schon vergeben worden ist.

Cursor an Position vor letztem Programm-Start bzw. Direct-Aufruf

==> <Alternate> + <7>

Cursor an erste Position bei Editor-Start

==> <Alternate> + <8>

Cursor an Position vor letztem Such-Start

==> <Alternate> + <9>

Cursor an zuletzt geänderte Position

==> <Alternate> + <0>

24.6 Prozedur-Liste

Seite:	
489	PROCEDURE access(ac.titel$,ac.exit$)
222	PROCEDURE astring(s.pos%,z.str$,s.str$,VAR bk$)
458	PROCEDURE atari(aptr1%,aptr2%,aptr3%,aptr4%,aptr5%)
462	PROCEDURE backup(pr$,ex$)
692	PROCEDURE blinker(b.flg%,bxl%,byo%,bxr%,byu%)
620	PROCEDURE boundary(flag%)
238	PROCEDURE cbox(mod%,xp%,yp%,rd%,wi%)
297	PROCEDURE clip(clxl%,clyo%,clxr%,clyu%)
298	PROCEDURE clip(windownummer%)
297	PROCEDURE clip_off
252	PROCEDURE ctri(mod%,xp%,yp%,rd%,wi%)
225	PROCEDURE cut(c.str$,c.sgn$,c.brt%,c.vec%)
695	PROCEDURE dcolor
441	PROCEDURE debbie
97	PROCEDURE degload(p.nm$,p.ad%,c.ad%,p.rs%)
99	PROCEDURE degsave(p.name$,p.adrs%)
256	PROCEDURE dfill(d.x%,d.y%,d.f%)
440	PROCEDURE distort(x1%,y1%,x2%,y2%,x%,y%,w%,p,s,v%,m%,sf%,ss,r%)
724	PROCEDURE dline(idx%)

24.7 Liste der IKB-KOMMANDOS

OUT 4,7 .... Maustasten-Reaktion einstellen.
OUT 4,8 .... Mauskontakt einschalten.
OUT 4,9 .... Absolute Mauskoordinaten liefern.
OUT 4,10 .... Maus-Bewegungen als Cursor-Tasten-Druck.
OUT 4,11 .... Raster für Mausmeldungen einstellen.
OUT 4,12 .... Maus-Skalierung.
OUT 4,13 .... Absolute Mausposition melden.
OUT 4,14 .... Koordinatenzähler setzen.
OUT 4,15 .... Pixel-Koordinate 0/0 liegt links unten.
OUT 4,16 .... Pixel-Koordinate 0/0 liegt links oben.
OUT 4,17 .... Kontrolle nach OUT 4,19 einschalten.
OUT 4,18 .... Mauskontakt ausschalten.
OUT 4,19 .... Tastatur-Kontrolle ausschalten.
OUT 4,20 .... Maus-Port 0 als Joystick-Port.
OUT 4,21 .... Joystick-Modus und Maus ausschalten.
OUT 4,22 .... 1 mal Port 0 im Joystick-Modus abfragen.
OUT 4,23 .... Joystick-Dauermeldung einschalten.
OUT 4,24 .... Firebutton-Dauermeldung einschalten.
OUT 4,25 .... Maus-Port 0 in Joystick-Modus schalten.
OUT 4,26 .... Joystick-Dauermeldung ausschalten.
OUT 4,27 .... Uhrzeit setzen.
OUT 4,28 .... Uhrzeit lesen.
OUT 4,29 .... Tastaturspeicher bestimmen und belegen.
OUT 4,30 .... Tastaturspeicher lesen.
OUT 4,31 .... Programm im Tastaturspeicher starten.

24.8 Liste der VT52-Escape-Sequenzen

CHR$(27)+"A" .... Cursor eine Zeile nach oben, ohne scrollen.
CHR$(27)+"B" .... Cursor eine Zeile nach unten, wenn.
CHR$(27)+"C" .... Cursor ein Zeichen nach rechts.
CHR$(27)+"D" .... Cursor ein Zeichen nach links.
CHR$(27)+"E" .... Clear screen/Cursor auf Home setzen.
CHR$(27)+"H" .... Cursor auf Home setzen.
CHR$(27)+"I" .... Cursor eine Zeile hoch + ggfs. scrollen.
CHR$(27)+"J" .... Bildschirm ab Cursor löschen.
CHR$(27)+"K" .... Zeile ab Cursor löschen.
CHR$(27)+"L" .... Zeile einfügen Rest 1 Zeile abwärts.
CHR$(27)+"M" .... Cursor-Zeile löschen und Rest nachziehen.
CHR$(27)+"Y"+CHR$(X+31)+CHR$(Y+31) .... Cursor positionieren.
CHR$(27)+"b"+CHR$(F) .... Cursor-Farbe F setzen.
CHR$(27)+"c"+CHR$(F) .... Zeichenhintergrundfarbe F setzen.
CHR$(27)+"d" .... Bildschirm bis Cursor löschen.
CHR$(27)+"e" .... Cursor anschalten.
CHR$(27)+"f" .... Cursor ausschalten.
CHR$(27)+"j" .... Position des Cursors speichern.
CHR$(27)+"k" .... Cursor auf gespeicherte Position setzen.
CHR$(27)+"l" .... Zeile löschen (Rest nicht nachziehen).
CHR$(27)+"o" .... Zeile bis Cursor löschen.
CHR$(27)+"p" .... Revers-Modus ein (weiß auf schwarz).
CHR$(27)+"q" .... Revers-Modus aus (schwarz auf weiß).
CHR$(27)+"v" .... Zeilen-Überlauf an.
CHR$(27)+"w" .... Zeilen-Überlauf aus.

24.9 Liste der System-Funktionen

BIOS-Funktionen

Var=BIOS(0,L:Adresse) .... -GETMPB-
Var=BIOS(1,Device) .... -BCONSTAT-
Var=BIOS(2,Device) .... -BCONIN-
Var=BIOS(3,Device) .... -BCONOUT-
Var=BIOS(4,Mod,L:Adr,Anz,Skt,Disk) .... -RWABS-
Var=BIOS(5,Exc_index,L:Adresse) .... -SETEXEC-
Var=BIOS(6) .... -TICKCAL-
Var=BIOS(7,Laufwerk) .... -GETBPB-
Var=BIOS(8,Device) .... -BCOSTAT-
Var=BIOS(9,Laufwerk) .... -MEDIACH-
Var=BIOS(10) .... -DRVMAP-
Var=BIOS(11,Status) .... -KBSHIFT-

GEMDOS-Funktionen

Var=GEMDOS(0) .... -PTERM-
Var=GEMDOS(1) .... -CONIN-
Var=GEMDOS(2,Ascii) .... -CONOUT-
Var=GEMDOS(3) .... -AUXIN-
Var=GEMDOS(4,Ascii) .... -AUXOUT-
Var=GEMDOS(5,Ascii) .... -PRNOUT-
Var=GEMDOS(6,Ascii) .... -RAWCONIO-
Var=GEMDOS(7) .... -RAWCONIN-
Var=GEMDOS(8) .... -RAWNECIN-
Var=GEMDOS(9,L:Adresse) .... -CONWS-
Var=GEMDOS(10,L:Adresse) .... -CONRS-
Var=GEMDOS(11) .... -CONSTAT-
Var=GEMDOS(14,Laufwerk) .... -SETDRV-
Var=GEMDOS(16) .... -CONOUTSTAT-
Var=GEMDOS(17) .... -PRTOUTSTAT-
Var=GEMDOS(18) .... -AUXINSTAT-
Var=GEMDOS(19) .... -AUXOUTSTAT-
Var=GEMDOS(25) .... -CURDRV-
Var=GEMDOS(26,L:Adresse) .... -SETDTA-
Var=GEMDOS(32,L:Adresse) .... -SUPER-
Var=GEMDOS(42) .... -GETDATE-
Var=GEMDOS(43,Format) .... -SETDATE-
Var=GEMDOS(44) .... -GETTIME-
Var=GEMDOS(45,Format) .... -SETTIME-
Var=GEMDOS(47) .... -GETDTA-
Var=GEMDOS(48) .... -GETVERS-
Var=GEMDOS(49,L:Byt,Wert) .... -KEEP PROCESS-
Var=GEMDOS(54,L:Adr,Laufw) .... -GETDFREE-
Var=GEMDOS(57,L:Adresse) .... -MKDIR-
Var=GEMDOS(58,L:Adresse) .... -RMDIR-
Var=GEMDOS(59,L:Adresse) .... -CHDIR-
Var=GEMDOS(60,L:Adresse,Attribut) .... -CREATE-
Var=GEMDOS(61,L:Adresse,Modus) .... -OPEN-
Var=GEMDOS(62,Kanal) .... -CLOSE-
Var=GEMDOS(63,Kanal,L:Länge,L:Adr) .... -READ-
Var=GEMDOS(64,Kanal,L:Länge,L:Adr) .... -WRITE-
Var=GEMDOS(65,L:Adresse) .... -UNLINK-
Var=GEMDOS(66,L:Bytes,Kanal,Modus) .... -LSEEK-

Var=GEMDOS(67,L:Adr,Modus,Attr) .... -CHMODE-
Var=GEMDOS(69,Kanal) .... -DUP-
Var=GEMDOS(70,Kanal,Standard) .... -FORCE-
Var=GEMDOS(71,L:Adresse,Laufwerk) .... -GETDIR-
Var=GEMDOS(72,L:Bytes) .... -MALLOC-
Var=GEMDOS(73,L:Adresse) .... -MFREE-
Var=GEMDOS(74,0,L:Adr,L:Bytes) .... -SETBLOCK-
Var=GEMDOS(75,Md,L:Nam,L:Com,L:Env) .... -PEXEC-
Var=GEMDOS(76,Wert) .... -PTERM-
Var=GEMDOS(78,L:Nam_adr,Attribut) .... -SFIRST-
Var=GEMDOS(79) .... -SNEXT-
Var=GEMDOS(86,L:Altadr,L:Neuadr,0) .... -RENAME-
Var=GEMDOS(87,L:Adr,Kanal,Modus) - .... GSDTOF-

XBIOS-Funktionen

Var=XBIOS(0,Typ,L:Adr1,L:Adr2) .... -INITMOUSE-
Var=XBIOS(1,Bytes) .... -SSBRK-
Var=XBIOS(2) .... -PHYSBASE-
Var=XBIOS(3) .... -LOGBASE-
Var=XBIOS(4) .... -GETREZ-
Var=XBIOS(5,L:Log,L:Phys,Res) .... -SETSCREEN-
Var=XBIOS(6,L:Adresse) .... -SETPALETTE-
Var=XBIOS(7,Register,Farbwert) .... -SETCOLOR-
Var=XBIOS(8,L:Adr,L:0,Disk,...
...Sekt,Trck,Sid,Anz) .... -FLOPRD-
Var=XBIOS(9,L:Adr,L:0,Disk,...
...Sekt,Trck,Sid,Anz) .... -FLOPWR-
Var=XBIOS(10,L:Adr,L:0,Disk,Anz,...
...Trck,Sid,Intrl,L:Mgc,Virg) .... -FLOPFMT-
Var=XBIOS(11) .... Kein Effekt!
Var=XBIOS(12,Byt,L:Adr) .... -MIDIWS-
Var=XBIOS(13,Mfp_index,L:Adr) .... -MFPINT-
Var=XBIOS(14,Device) .... -IOREC-
Var=XBIOS(15,Ba,Md,Usr,Rsr,Csr,Scr) .... -RSCONF-
Var=XBIOS(16,L:Nrm,L:Shft,L:Caps) .... -KEYTABLE-
Var=XBIOS(17) .... -RANDOM-
Var=XBIOS(18,L:Ad,L:0,L:Se,Tp,Fl) .... -PROTOBT-
Var=XBIOS(19,L:Adr,L:0,Disk,...
...Sekt,Trck,Sid,Anz) .... -FLOPVER-
Var=XBIOS(20) .... -SCRDMP-
Var=XBIOS(21,Modus,Frequenz) .... -CURSCONF-
Var=XBIOS(22,L:Adresse) .... -SETTIME-
Var=XBIOS(23) .... -GETTIME-
Var=XBIOS(24) .... -BIOSKEYS-
Var=XBIOS(25,Bytes,L:Adresse) .... -IKBDWS-
Var=XBIOS(26,Mfp_index) .... -JDISINT-
Var=XBIOS(27,Mfp_index) .... -JENABIN-
Var=XBIOS(28,Byte,Register) .... -GIACCES-
Var=XBIOS(29,Vektor) .... -OFFGIBIT-
Var=XBIOS(30,Vektor) .... -ONGIBIT-
Var=XBIOS(31,Tim,Cntrl,Dat,L:Adr) .... -XBTIMER-
Var=XBIOS(32,L:Adresse) .... -DOSOUND-
Var=XBIOS(33,Modus) .... -SETPRT-
Var=XBIOS(34,Modus) .... -KBDVBASE-
Var=XBIOS(35,Delay,Repeat) .... -KBRATE-
Var=XBIOS(36,L:Adresse) .... -PRTBLK-

Var=XBIOS(37) .... -VSYNC-
Var=XBIOS(38,L:Adresse) .... -SUPEXEC-
Var=XBIOS(39) .... -PUNTAES-

24.10 Thematische Übersicht

Ein-/Ausgabebefehle

Dateneingabe

FORM INPUT .... Formatierte String-Eingabe.
FORM INPUT AS .... Formatierte String-Eingabe m. Vorgabe.
INKEY$ .... Einzelzeichen von Tastatur holen.
INPUT .... Dateneingabe
INPUT$() .... Zeichenketteneingabe
LINE INPUT .... Zeichenketteneingabe

Datenausgabe

PRINT .... Daten ausgeben.
PRINT USING .... Daten formatiert ausgeben.
WRITE .... Daten ausgeben.

Bildschirmoperationen

HTAB .... Aktuelle Cursor-Spalte bestimmen.
LOCATE .... Cursor positionieren.
POS() .... Cursor-Spalte ermitteln.
SPC() .... Leerzeichen ausgeben.
TAB() .... Tabulator setzen.
VTAB .... Aktuelle Cursor-Zeile bestimmen.

Diskettenoperationen

BLOAD .... Datei in Speicherbereich laden.
BSAVE .... Speicherbereich auf Disk speichern.
CHAIN .... Programm Laden (Autostart).
CHDIR .... Ordner wechseln.
CHDRIVE .... Aktuelles Laufwerk bestimmen.
DFREE() .... Freien Disketten-Speicherplatz ausgeben.
DIR .... Directory ausgeben.
DIR$() .... Aktuellen Ordnernamen ermitteln.
EXIST() .... Existenz einer Datei prüfen.
FGETDTA() .... Disk-Transfer-Adresse ermitteln.
FILES .... Directory (erweitert) ausgeben.
FSETDTA() .... Disk-Transfer-Adresse bestimmen.
FSFIRST() .... Datei suchen.
FSNEXT() .... Weitere Datei suchen.
KILL .... Disk-Datei löschen.
LIST .... Programm listen/speichern (ASCII).
LOAD .... Programm in Abeitsspeicher laden.
MKDIR .... Ordner erzeugen.
NAME .... Datei umbenennen.
PSAVE .... Programm speichern (listgeschützt).

RENAME .... Datei umbenennen.
RMDIR .... Ordner löschen.
SAVE .... Programm speichern (codiert).

Dateihandhabung

BGET .... Teildatei lesen.
BPUT .... Teildatei schreiben.
CLOSE .... Datenkanal schließen.
EOF() .... Datei auf Dateiende prüfen.
FIELD .... Datensatz in Felder unterteilen.
GET # .... Datensatz lesen.
LOC() .... File-Pointer-Position.
LOF() .... Dateilänge ermitteln.
OPEN .... Datenkanal öffnen.
PUT # .... Datensatz schreiben.
RECALL .... String-Feld aus Datei lesen.
RECORD .... Satz-Pointer für GET#/PUT# setzen.
RELSEEK .... File-Pointer verschieben.
SEEK .... File-Pointer setzen.
TOUCH .... Datei-Zeiteintrag aktualisieren.
STORE .... String-Feld in Datei ablegen.

Port-Ein-/-Ausgabe-Befehle

INP? .... Port auf Empfangsbereitschaft testen.
INP .... Daten byteweise von Peripherie lesen.
INPAUX$ .... Text-String vom seriellen Port (RS232) lesen.
INPMID$ .... Text-String vom MID-IN-Port lesen.
OUT? .... Port auf Ausgabebereitschaft testen.
OUT .... Daten byteweise an Peripherie ausgeben.

Druckeranweisungen

HARDCOPY .... Bildschirm auf Drucker ausgeben.
LLIST .... Programm-Listing ausdrucken.
LPOS() .... Druckkopfposition ermitteln.
LPRINT .... Daten an Drucker ausgeben.

Sound-Chip-Anweisungen

SOUND .... Klangausgabe
WAVE .... Sound-Attribute einstellen.

Programmstruktur

Schleifenkonstruktionen

DO ... LOOP .... Endlosschleife
DO [WHILE Bed] [UNTIL Bed]
... . Doppelt bedingte Schleife (V3.0)
LOOP [WHILE Bed] [UNTIL Bed]
FOR ... NEXT .... Zählschleife
REPEAT ... UNTIL .... Bedingte Schleife.

WHILE ... WEND .... Bedingte Schleife.

Bedingte Verzweigungen

EXIT IF .... Bedingter Schleifenabbruch.
IF [ELSE] ENDIF .... Bedingungsabfrage
ELSE IF .... Unter-Bedingungsabfrage (V3.0).
SELECT [CONT] CASE[TO] [DEFAULT]ENDSELECT .... Fall-Abfrage

Bereichsdeklarationen

! .... Kommentar innerhalb einer Befehlszeile.
DATA .... Daten-Speicher deklarieren.
READ .... DATA-Werte auslesen.
REM .... Kommentar einfügen.
RESTORE .... DATA-Zeiger setzen.

Variablendeklarationen

DEFBIT .... Boole-Variable(n) deklarieren.
DEFBYT .... 1-Byte-Integervariablen deklarieren.
DEFFLT .... 8-Byte-Fließkommavariablen deklarieren.
DEFINT .... 4-Byte-Integervariablen deklarieren.
DEFSTR .... Zeichenkettenvariable(n) deklarieren.
DEFWRD .... 2-Byte-Integervariablen deklarieren.

Unterprogramme

DEFFN .... Funktion definieren.
FN .... Funktion aufrufen.
FUNCTION..RETURN..ENDFUNC .... Funktion
GOSUB .... Verzweigung zu einer PROCEDURE.
GOTO .... Unbedingter Sprung zu einem Label.
LOCAL .... Lokale Variablen deklarieren.
ON ... GOSUB .... Bedingte Verzweigung zu Prozeduren.
ON BREAK [CONT] [GOSUB] .... Break-Funktion behandeln.
PROCEDURE ...RETURN .... Prozedur-Titel
VAR .... Direkte Variablen-Übergabe.

Assembler-/C-/PRG-Programmaufrufe

C:() .... Maschinenprogramm (C-compiliert) aufrufen.
CALL .... Maschinenprogramm (assembliert) aufrufen.
EXEC .... PRG/.TOS/.TTP-Programm laden/starten.
MONITOR .... Maschinen-Programm aufrufen.
RCALL .... Masch.-Programmaufruf m. Registerzugriff.

Textoperationen

Stringmanipulationen

- **MID$()=** .... **Teil-String zuweisen.**
- **LSET** .... **Zeichen(kette) linksbündig einsetzen.**
- **RSET** .... **Zeichen(kette) rechtsbündig einsetzen.**

String-Analyse

- **INSTR()** .... **Zeichen(kette) in einem String suchen.**
- **LEFT$()** .... **Linksbündigen Teil-String ermitteln.**
- **LEN()** .... **String-Länge ermitteln.**
- **MID$()** .... **Beliebigen Teil-String ermitteln.**
- **PRED()** .... **Nächstkleineres ASCII-Zeichen ermitteln.**
- **RIGHT$()** .... **Rechtsbündigen Teil-String ermitteln.**
- **RINSTR()** .... **Zeichen(kette) in String rückwärts suchen.**
- **SUCC()** .... **Nächstgrößeres ASCII-Zeichen ermitteln.**

String-Formatierung

- **SPACE$()** .... **Leerzeichen-String bilden.**
- **STRING$()** .... **Mehrfach-Zeichenkette bilden.**
- **TRIM$()** .... **Space-Zeichen eliminieren.**
- **UPPER$()** .... **Buchstabenumwandlung Klein => Groß.**

Arithmetikbefehle

Mathematische Operationen

- **ADD** .... **Additionsbefehl**
- **DEC** .... **Dekrementierung**
- **DIV** .... **Divisionsbefehl**
- **INC** .... **Inkrementierung**
- **MUL** .... **Multiplikationsbefehl**
- **SUB** .... **Subtraktionsbefehl**
- **ADD()** .... **Integer-Additionsfunktion**
- **DIV()** .... **Integer-Divisionsfunktion**
- **MOD()** .... **Integer-Modula-Funktion**
- **MUL()** .... **Integer-Multiplikationsfunktion**
- **SUB()** .... **Integer-Subtraktionsfunktion**

Numerische Funktionen

- **ABS()** .... **Betrags-Funktion**
- **EVEN()** .... **Zahl auf "gerade" testen.**
- **EXP()** .... **Exponential-Funktion**
- **FIX()** .... **Ganzzahl-Funktion**
- **FRAC()** .... **Dezimalstellen-Funktion**
- **INT()** .... **Ganzzahl-Funktion**
- **LOG()** .... **Logarithmus-Funktion**
- **ODD()** .... **Zahl auf "ungerade" testen.**
- **PRED()** .... **Nächstkleinere Ganzzahl ermitteln.**
- **ROUND()** .... **Rundungs-Funktion**

SGN() .... Vorzeichen ermitteln
SQR() .... Wurzel-Funktion
SUCC() .... Nächstgrößere Ganzzahl ermitteln.
TRUNC() .... Ganzzahl-Funktion

Trigonometrische Funktionen

ACOS() .... Arcuscosinus-Funktion
ASIN() .... Arcussinus-Funktion
ATN() .... Arcustangens-Funktion
COS() .... Cosinus-Funktion
COSQ() .... Interpolierte Cosinus-Funktion mit Grad-Angabe.
DEG() .... Umwandlung in Grad
PI .... Kreiszahl
RAD() .... Umwandlung in Radian (Bogenmaß).
SIN() .... Sinus-Funktion
SINQ() .... Interpolierte Sinus-Funktion mit Grad-Angabe.
TAN() .... Tangens-Funktion

Vergleichsoperationen

MAX() .... Größten Wert/String ermitteln.
MIN() .... Kleinsten Wert/String ermitteln.

Bit-Operationen

AND() .... Konjunktions-Funktion
BCHG() .... Einzel-Bit wechseln (Xor-en).
BCLR() .... Einzel-Bit löschen.
BSET() .... Einzel-Bit setzen.
BTST() .... Einzel-Bit auf an/aus testen.
BYTE() .... Vorzeichenloses LO-Byte eines Wertes liefern.
CARD() .... Vorzeichenloses LO-Word eines Wertes liefern.
EQV() .... Äquivalenz-Funktion
IMP() .... Implikations-Funktion
OR() .... Disjunktions-Funktion
SHL() .... Bits links verschieben.
SHR() .... Bits logisch rechts verschieben.
ROL() .... Bits links rotieren.
ROR() .... Bits rechts rotieren.
SWAP() .... HI- und LO-Word vertauschen.
WORD() .... Wert auf 32 Bit erweitern.
XOR() .... eXclusivOR-Funktion.

Zufallswerterzeugung

RAND() 16-Bit-Integer-Zufallszahl
RANDOM() 32-Bit-Integer-Zufallszahl
RANDOMIZE .... Zufallszahlengenerator initialisieren.
RND() .... Dezimalstellen-Zufallszahl

Grafik

Grafikdefinitionen

BOUNDARY .... P-Grafikumrandung an/aus.
COLOR .... Linienfarbe bestimmen.
DEFFILL .... Füllmuster bestimmen.
DEFLINE .... Linien-Modi bestimmen.
DEFMARK .... Markierungs-Symbol bestimmen.
DEFMOUSE .... Mausform bestimmen.
DEFTEXT .... Grafik-Text-Modi bestimmen.
GRAPHMODE .... Grafikmodus bestimmen.
SETCOLOR .... Hardware-Farbregister einstellen.
VSETCOLOR .... VDI-Farbregister einstellen.

Objektgrafikbefehle

BOX, PBOX .... Rechteck zeichnen.
CIRCLE, PCIRCLE .... Kreis(bogen) zeichnen.
ELLIPSE, PELLIPSE .... Ellipse(nbogen) zeichnen.
FILL .... Flächen mit Muster füllen.
POLYFILL .... Polygon zeichnen, gefüllt.
POLYMARK .... Polygon-Eckpunkte markieren.
RBOX, PRBOX .... Rechteck abgerundet zeichnen.
TEXT .... Text im Grafikmodus ausgeben.

Strich-/Punktgrafik

DRAW .... Punkte zeichnen und verbinden.
DRAW $ .... Plotter-(Turtle-)Grafik
DRAW() .... Plotter-(Turtle-)Attribute liefern.
LINE .... Linie zeichnen.
PLOT .... Punkt zeichnen.
POINT() .... Bildschirmpunkt-Farbwert ermitteln.
POLYLINE .... Polygon zeichnen.
SETDRAW .... DRAW-Turtle positionieren.

Line-A-Grafikbefehle

ACHAR .... Line-A-Einzel-Textzeichen ausgeben.
ACLIP .... Line-A-Clipbox setzen.
ALINE .... Line-A-Linie zeichnen.
APOLY .... Line-A-Vieleck zeichnen.
ARECT .... Line-A: Gefülltes Rechteck zeichnen.
ATEXT .... Line-A-Grafiktext ausgeben.
HLINE .... Line-A: Horizontale Linie zeichnen.
PSET .... Line-A-Punkt zeichnen.
PTST .... Line-A-Bildschirmpunkt testen.

Grafikoperationen

CLIP .... Grafikausgabe begrenzen/Nullpunkt setzen.
GET .... Bildschirmbereich speichern.
HIDEM .... Mauszeiger ausschalten.
PUT .... Bildschirmbereich setzen.

RC_COPY .... Speicherinternes Rechteck-Copy.
RC_INTERSECT() .... Überlappung zweier Rechtecke.
SGET .... Bildschirm puffern.
SHOWM .... Mauszeiger anschalten.
SPRITE .... Sprite setzen und löschen.
SPUT .... Bildschirm setzen.
VSYNC .... VBL-Synchronisation.

Datenumwandlung

ASC() .... Textzeichen => ASCII-Wert.
BIN$() .... Numerisch => Binär.
CFLOAT() .... Integerwert => Fließkommawert.
CHR$() .... ASCII => Textzeichen.
CINT() .... Fließkommawert => Integerwert.
CVI(),CVL(),CVS(),CVF(),CVD() .... String => Format-Zahl.
HEX$() .... Numerisch => Hexadezimal.
MKI$(),MKL$(),MKS$(),MKF$(),MKD$() .... Format-Zahl=>String.
OCT$() .... Numerisch => Oktal.
STR$() .... Numerisch => String.
VAL() .... String => Numerisch.
VAL?() .... Anzahl wandelbarer Textzeichen ermitteln.

Feld-/Speicher- und Zeigeroperationen

Feldoperationen

ARRAYFILL .... Feld mit Wert belegen.
DELETE .... Einzelelement aus Feld löschen.
DIM .... Feld(er) dimensionieren.
DIM?() .... Menge der Feldelemente ermitteln.
ERASE .... Feld(er) löschen.
INSERT .... Einzelelement in Feld einfügen.
OPTION BASE .... Feld-Basiselement bestimmen.
QSORT .... Feld (-Bereich) Quick-Sortierung.
SSORT .... Feld (-Bereich) Shell-Sortierung.

Speicheroperationen

ABSOLUTE .... Variable auf Adresse setzen.
BITBLT .... Speicherbereiche verknüpfen.
BMOVE .... Speicherblock kopieren.
BYTE{}, CARD{}, LONG{} .... Speicherinhalt lesen (User-Modus).
CHAR{} .... C-Text lesen.
CHAR{}= .... C-Text schreiben.
DOUBLE{}, SINGLE{} .... IEEE-Double/Single-Realformat lesen.
DOUBLE{}=, SINGLE{}= .... IEEE-Realformat schreiben.
FLOAT{} .... 8 Byte in GFA-3.0-BASIC-Realformat lesen.
FLOAT{}= .... 8 Byte in GFA-3.0-BASIC-Realformat schreiben.
INT{} / WORD{} .... 2 Byte als Vorzeichen-Integer lesen.
INT{}= / WORD{}= .... 2 Byte Vorzeichen-Integer schreiben.
PEEK, DPEEK, LPEEK .... Speicher lesen (Supervisor-Modus).
POKE,DPOKE,LPOKE .... Speicherinhalt ändern(User-Modus).
SPOKE,SDPOKE,SLPOKE .... Supervisorpoke

Speicherverwaltung

INLINE .... BASIC-interne Speicherreservierung.
MALLOC() .... System-Speicher-Reservierung.
MFREE() .... MALLOC()-Speicher freigeben.
MSHRINK() .... MALLOC()-Speicher einschränken.
RESERVE .... BASIC-Arbeitsspeicher festlegen.

Zeigeroperationen

***** .... Variablen-Pointer
ARRPTR .... String-/Feld-Descriptor-Adresse ermitteln.
VARPTR .... Variablen-Adresse ermitteln.

Programmkontrolle

Programmstart und -ende

CONT .... Programm (nach STOP-Befehl) fortsetzen.
EDIT .... Programm beenden.
END .... Programm beenden.
QUIT .... Programmende (Rückkehr zum Desktop).
RUN .... Programm starten.
STOP .... Programm unterbrechen.
SYSTEM .... Programmende (Interpreter verlassen).

Löschfunktionen

CLEAR .... Felder und Variablen löschen.
CLR .... Einzelvariablen löschen.
CLS .... Bildschirm löschen.
NEW .... Programmspeicher löschen.

Zeitoperationen

DATE$ .... Systemdatum ermitteln.
DELAY .... 1/1-Sek.-Wartefunktion.
PAUSE .... 1/50-Sek.-Wartefunktion.
SETTIME .... Uhrzeit und Datum einstellen.
TIME$ .... System-Uhrzeit ermitteln.
TIMER .... Laufzeit ermitteln.

Fehlerbehandlung

ERR .... Fehler-Code ermitteln.
ERR$ .... Fehlertext liefern.
ERROR .... Fehler simulieren.
FATAL .... Fehlerart ermitteln.
ON ERROR [GOSUB] .... Verzweigung bei Fehler.
RESUME .... Programm nach Error-Routine fortsetzen.

Auskünfte

- **BASEPAGE**....Aktuelle Basepage-Adresse liefern.
- **CRSCOL**....Aktuelle Cursor-Spalte liefern.
- **CRSLIN**....Aktuelle Cursor-Zeile liefern.
- **FRE()**....Freien Speicherplatz ermitteln.
- **HIMEM**....Erstes Byte hinter BASIC-Speicher liefern.
- **L-A**....Basis-Adresse der Line-A-Variablen liefern.
- **TYPE()**....Variablentyp ermitteln.
- **WORK_OUT()**....Open Workstation-Rückgabe-Array.

Tastaturkontrolle

- **KEYDEF**....Funktionstasten belegen.
- **KEYGET**....Auf Taste warten.
- **KEYLOOK**....1. Zeichen aus Tastaturpuffer lesen.
- **KEYPAD**....Tastatur-Attribute definieren.
- **KEYPRESS**....Tastendruck simulieren.
- **KEYTEST**....Tastatur abfragen.

Multitasking

- **AFTER x GOSUB**....Single-Interruptroutinenaufruf.
- **AFTER CONT**....Single-Interruptroutine freigeben.
- **AFTER STOP**....Single-Interruptroutine sperren.
- **EVERY x GOSUB**....Interrupt-Routinenaufruf.
- **EVERY CONT**....Interrupt-Routine freigeben.
- **EVERY STO**....Interrupt-Routine sperren.

Debugging

- **DUMP**....Variableninhalte/Namen ausgeben.
- **TRACE$**....Im Trace-Modus aktuelle Befehlszeile liefern.
- **TROFF**....Trace-Modus ausschalten.
- **TRON**....Trace-Modus einschalten.
- **TRON Proc**....Trace-Modus in Prozedur lenken.

Diverses

- **$**....Textbereich für V3.0-Compiler deklarieren.
- **DEFLIST**....Listing-Format festlegen.
- **DEFNUM**....Rundung von Ziffern-Ausgaben.
- **FALSE**....Unwahr-Konstante
- **LET**....Daten zuweisen.
- **MODE**....Zahlen/Datum -> Europa/USA wählen.
- **OPTION**....Compiler-Steuerung
- **TRUE**....Wahr-Konstante
- **SWAP**....Variablen/Felder/Pointer tauschen.
- **VOID**....Dummy-Zuweisung

Interaktionen (Programm/Benutzer)

- **ALERT**....Alert-Box erstellen.
- **FILESELECT**....Datei auswählen.
- **MOUSE**....Maus-Status ermitteln (gesamt).

Window-Programmierung mit BASIC-Befehlen

Menüprogrammierung mit BASIC-Befehlen

Ereignisüberwachung mit BASIC-Befehlen

GDOS/VDI-Bibliothek

AES-Bibliothek

Application-Manager

APPL_EXIT() .... GEM-Applikation abmelden.
APPL_FIND() .... Applikations-Identifikator ermitteln.
APPL_INIT()... GEM-Applikation anmelden.
APPL_READ() ....Aus Ereignispuffer lesen.
APPL_TPLAY() ....Wiedergabe gespeicherter Ereignisse.
APPL_TRECORD() ....Ereignisse speichern.
APPL_WRITE()... In Ereignispuffer schreiben.

Event-Manager

EVNT_BUTTON() .... Auf Maustasten-Ereignis warten.
EVNT_DCLICK()..........................Doppelklick-Geschwindigkeit einstellen.
EVNT_KEYBD().. Auf Tastatur-Ereignis warten.
EVNT_MESAG()...................................Auf Eventpuffer-Ereignis warten.
EVNT_MOUSE()................................Auf Mauspositions-Ereignis warten.
EVNT_MULTI() .... Auf Mehrfach-Ereignis warten.
EVNT_TIMER()...Auf Zeit-Ereignis warten.

Formular-Manager

FORM_ALERT()..........................Hinweis-Formular (Alert-Box) erzeugen.
FORM_BUTTON().................................... Mausüberwachung im Formular.
FORM_CENTER() .... Formularkoordinaten zentrieren.
FORM_DIAL() ....Formular-Hintergrund puffern/restaurieren.
FORM_DO()..................................GEM übernimmt Formular-Verwaltung.
FORM_ERROR() ....Hinweis-Formular (TOS-Error) erzeugen.
FORM_KEYBD() ....Formular-Texteingabe

Fileselect-Manager

FSEL_INPUT()... Fileselect-Box produzieren.
GRAPHICS-MANAGER...................................... Grafikeffekte überwachen.
GRAF_DRAGBOX() .... Schiebebox produzieren.
GRAF_GROWBOX() .... Box-Vergrößerungseffekt produzieren.
GRAF_HANDLE() .... VDI-Handle u. Zeichenmaße ermitteln.
GRAF_MKSTATE()....................Umschalttasten-/Maus-Status ermitteln.
GRAF_MOUSE()............................. Mausform bestimmen (DEFMOUSE).
GRAF_MOVEBOX() ....Box-Bewegungseffekt produzieren.
GRAF_RUBBERBOX()...................Gummiband-Box (Lasso) produzieren.
GRAF_SHRINKBOX() ....Box-Verkleinerungseffekt produzieren.
GRAF_SLIDEBOX() ....Schiebebox innerhalb von Formularen.
GRAF_WATCHBOX() .... Objektstatus gemäß Mausposition.

Menü-Manager

MENU_BAR() .... Zeichnen/Löschen einer Menüleiste.
MENU_ICHECK() .... Menü-Checkmark zeichnen/löschen.
MENU_IENABLE() ....Menüeinträge aktivieren/deaktivieren.
MENU_TEXT() .... Menü-Text anpassen.
MENU_TNORMAL()..Menü-Titel invers/normal.
MENU_REGISTER() .... Accessory in Register eintragen.

Objekt-Manager

OBJC_ADD() .... Objekt in Baum einfügen.
OBJC_CHANGE() .... Objekt-Status ändern.
OBJC_DELETE() .... Objekt deaktivieren.
OBJC_DRAW() .... Objekt(e) darstellen.
OBJC_EDIT() .... Objekt-Texteingabe
OBJC_FIND() .... Objektnummer ermitteln.
OBJC_OFFSET() .... Absolute Objektposition ermitteln.
OBJC_ORDER() .... Objekt neu zuordnen.

Resource-Manager

RSRC_FREE() .... Resource-Speicher freigeben.
RSRC_GADDR() .... Resource-Adresse ermitteln.
RSRC_LOAD() .... Resource-Datei laden.
RSRC_OBFIX() .... Objektkoordinaten umwandeln.
RSRC_SADDR() .... Resource-Adresse einfügen.

Scrap-Manager

SCRP_READ() .... Globalen GEM-Puffer (Clipboard) lesen.
SCRP_WRITE() .... Globalen GEM-Puffer (Clipboard) schreiben.

Shell-Manager

SHEL_ENVRN() .... DOS-Environment-Adresse ermitteln.
SHEL_FIND() .... Applikationsnamen auf Disk suchen.
SHEL_GET() .... Environment-Puffer lesen.
SHEL_PUT() .... In Environment-Puffer schreiben.
SHEL_READ() .... Applikationsnamen und -kommandozeile lesen.
SHEL_WRITE() .... Applikation anmelden/starten.

Window-Manager

WIND_CALC() .... Fensterkoordinaten und -maße ermitteln.
WIND_CLOSE() .... Fenster schließen (CLOSEW).
WIND_CREATE() .... Fenster definieren/anmelden.
WIND_DELETE() .... Fenster löschen/abmelden.
WIND_FIND() .... Fenster-Handle ermitteln.
WIND_GET() .... Fenster-Attribute ermitteln.
WIND_OPEN() .... Fenster darstellen (OPENW).
WIND_SET() .... Fenster-Attribute ändern.
WIND_UPDATE() .... Fenster-Darstellung kontrollieren.

Objektvariablen

OB_ADR() .... Objekt-Adresse
OB_NEXT() .... Nächstes Objekt derselben Ebene lesen.
OB_NEXT()= .... Nächstes Objekt derselben Ebene schreiben.
OB_HEAD() .... Erstes Objekt der Kind-Ebene lesen.
OB_HEAD()= .... Erstes Objekt der Kind-Ebene schreiben.
OB_TAIL() .... Letztes Objekt der Kind-Ebene lesen.
OB_TAIL()= .... Letztes Objekt der Kind-Ebene schreiben.
OB_TYPE() .... Objekt-Typ lesen.

OB_TYPE()= .... Objekt-Typ schreiben.
OB_FLAGS() .... Objekt-Attribute lesen.
OB_FLAGS()= .... Objekt-Attribute schreiben.
OB_STATE() .... Objekt-Status lesen.
OB_STATE()= .... Objekt-Status schreiben.
OB_SPEC() .... Adresse der Unter-Struktur lesen.
OB_SPEC()= .... Adresse der Unter-Struktur schreiben.
OB_X() .... Relative Objekt-X-Koordinate lesen.
OB_X()= .... Relative Objekt-X-Koordinate schreiben.
OB_Y() .... Relative Objekt-Y-Koordinate lesen.
OB_Y()= .... Relative Objekt-Y-Koordinate schreiben.
OB_W() .... Objektbreite lesen.
OB_W()= .... Objektbreite schreiben.
OB_H() .... Objekthöhe lesen.
OB_H()= .... Objekthöhe schreiben.

GEM-Adressen/-Felder

ADDRIN .... AES-Adreß-Input-Block
ADDRIN() .... Aus AES-ADDRIN-Block als Array lesen.
ADDRIN()= .... In AES-ADDRIN-Block als Array schreiben.
ADDROUT .... AES-Adreß-Output-Block
ADDROUT() .... Aus AES-ADDROUT-Block als Array lesen.
ADDROUT()= .... In AES-ADDROUT-Block als Array schreiben.
CONTRL .... VDI-Kontroll-Block
CONTRL() .... Aus VDI-CONTRL-Block als Array lesen.
CONTRL()= .... In VDI-CONTRL-Block als Array schreiben.
GB .... Startadresse des Array-Zeigerblocks.
GCONTRL .... AES-Kontroll-Block
GCONTRL() .... Aus AES-CONTRL-Block als Array lesen.
GCONTRL()= .... In AES-CONTRL-Block als Array schreiben.
GINTIN .... AES-Integer-Input-Block
GINTIN() .... Aus AES-INTIN-Block als Array lesen.
GINTIN()= .... In AES-INTIN-Block als Array schreiben.
GINTOUT .... AES-Integer-Output-Block
GINTOUT() .... Aus AES-INTOUT-Block als Array lesen.
GINTOUT()= .... In AES-INTOUT-Block als Array schreiben.
INTIN .... VDI-Integer-Input-Block
INTIN() .... Aus VDI-INTIN-Block als Array lesen.
INTIN()= .... In VDI-INTIN-Block als Array schreiben.
INTOUT .... VDI-Integer-Output-Block
INTOUT() .... Aus VDI-INTOUT-Block als Array lesen.
INTOUT()= .... In VDI-INTOUT-Block als Array schreiben.
PTSIN .... VDI-Punkt-Input-Block
PTSIN() .... Aus VDI-PTSIN-Block als Array lesen.
PTSIN()= .... In VDI-PTSIN-Block als Array schreiben.
PTSOUT .... VDI-Punkt-Output-Block
PTSOUT() .... Aus VDI-PTSOUT-Block als Array lesen.
PTSOUT()= .... In VDI-PTSOUT-Block als Array schreiben.
VDIBASE .... VDI-Verwaltungs-Block

Systemaufrufe

BIOS() .... BIOS-Routinen aufrufen.
GEMDOS() .... GEMDOS-Routinen aufrufen.
GEMSYS .... AES-Routinen aufrufen.
VDISYS .... VDI-Routinen aufrufen.
XBIOS() .... XBIOS-Routinen aufrufen.

24.11 ASCII-TABELLE

Standard-ASCII's

Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen
0 =		1 =	⇧	2 =	⇩	3 =	⇨	4 =	⇦	5 =	⊠	6 =	◩	7 =	⊠
8 =	✓	9 =	◷	10 =	🔔	11 =	♪	12 =	FF	13 =	CR	14 =	/|	15 =	|\
16 =	0	17 =	1	18 =	2	19 =	3	20 =	4	21 =	5	22 =	6	23 =	7
24 =	8	25 =	9	26 =	ə	27 =	ES	28 =	c	29 =	ɔ	30 =	x	31 =	ɏ
32 =		33 =	!	34 =	"	35 =	#	36 =	$	37 =	%	38 =	&	39 =	'
40 =	(	41 =	)	42 =	*	43 =	+	44 =	,	45 =	-	46 =	.	47 =	/
48 =	0	49 =	1	50 =	2	51 =	3	52 =	4	53 =	5	54 =	6	55 =	7
56 =	8	57 =	9	58 =	:	59 =	;	60 =	<	61 =	=	62 =	>	63 =	?
64 =	@	65 =	A	66 =	B	67 =	C	68 =	D	69 =	E	70 =	F	71 =	G
72 =	H	73 =	I	74 =	J	75 =	K	76 =	L	77 =	M	78 =	N	79 =	O
80 =	P	81 =	Q	82 =	R	83 =	S	84 =	T	85 =	U	86 =	V	87 =	W
88 =	X	89 =	Y	90 =	Z	91 =	[	92 =	\	93 =	]	94 =	^	95 =	_
96 =	`	97 =	a	98 =	b	99 =	c	100 =	d	101 =	e	102 =	f	103 =	g
104 =	h	105 =	i	106 =	j	107 =	k	108 =	l	109 =	m	110 =	n	111 =	o
112 =	p	113 =	q	114 =	r	115 =	s	116 =	t	117 =	u	118 =	v	119 =	w
120 =	x	121 =	y	122 =	z	123 =	{	124 =	|	125 =	}	126 =	~	127 =	Δ
128 =	Ç	129 =	ü	130 =	é	131 =	â	132 =	ä	133 =	à	134 =	å	135 =	ç
136 =	ê	137 =	ë	138 =	è	139 =	ï	140 =	î	141 =	ì	142 =	Ä	143 =	Å
144 =	É	145 =	æ	146 =	Æ	147 =	ô	148 =	ö	149 =	ò	150 =	û	151 =	ù
152 =	ÿ	153 =	Ö	154 =	Ü	155 =	¢	156 =	£	157 =	¥	158 =	ß	159 =	ƒ
160 =	á	161 =	í	162 =	ó	163 =	ú	164 =	ñ	165 =	Ñ	166 =	ª	167 =	º
168 =	¿	169 =	⌐	170 =	¬	171 =	½	172 =	¼	173 =	¡	174 =	«	175 =	»

Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen
176 =	ã	177 =	õ	178 =	Ø	179 =	ø	180 =	œ	181 =	Œ	182 =	À	183 =	Ã
184 =	Õ	185 =	¨	186 =	´	187 =	†	188 =	¶	189 =	©	190 =	®	191 =	™
192 =	ĳ	193 =	Ĳ	194 =	א	195 =	ב	196 =	ג	197 =	ד	198 =	ה	199 =	ו
200 =	ז	201 =	ח	202 =	ט	203 =	י	204 =	כ	205 =	ל	206 =	מ	207 =	נ
208 =	ס	209 =	ע	210 =	פ	211 =	צ	212 =	ק	213 =	ר	214 =	ש	215 =	ת
216 =	ן	217 =	ך	218 =	ם	219 =	ף	220 =	ץ	221 =	§	222 =	∧	223 =	∞
224 =	α	225 =	β	226 =	Γ	227 =	π	228 =	Σ	229 =	σ	230 =	µ	231 =	τ
232 =	Φ	233 =	Θ	234 =	Ω	235 =	δ	236 =	∮	237 =	ϕ	238 =	∈	239 =	∩
240 =	≡	241 =	±	242 =	≥	243 =	≤	244 =	⌠	245 =	⌡	246 =	÷	247 =	≈
248 =	°	249 =	•	250 =	·	251 =	√	252 =	ⁿ	253 =	²	254 =	³	255 =	¯

STEUERZEICHEN

Code	Zeichen	Name	Code	Zeichen	Name	Code	Zeichen	Name	Code	Zeichen	Name
0		NUL	1	⇧	SOH	2	⇩	STX	3	⇨	ETX
4	⇦	EOT	5	⊠	ENQ	6	◩	ACK	7	⊠	BEL
8	✓	BS	9	◷	HT	10	🔔	LF	11	♪	VT
12	FF	FF	13	CR	CR	14	/|	SO	15	|\	SI
16	0	DLE	17	1	DC1	18	2	DC2	19	3	DC3
20	4	DC4	21	5	NAK	22	6	SYN	23	7	ETB
24	8	CAN	25	9	EM	26	ə	SUB	27	ES	ESC
28	c	FS	29	ɔ	GS	30	x	RS	31	ɏ	US

24.12 GFA-BASIC-Fehlerliste

Editor-Fehlermeldungen

CASE OHNE SELECT -> Nur in V3.0
DO OHNE LOOP
ELSE OHNE IF
ELSE OHNE ENDIF
ENDFUNC OHNE FUNCTION -> Nur in V3.0
ENDIF OHNE IF
EXIT OHNE SCHLEIFE
FOR OHNE NEXT
FUNCTION DOPPELT DEFINIERT
FUNCTION IN SCHLEIFE -> Nur in V3.0
FUNCTION OHNE ENDFUNC -> Nur in V3.0
GOTO IN/AUS FOR-NEXT ODER PROCEDURE
IF OHNE ENDIF
KEIN RESUME IN FUNCTION -> Nur in V3.0
LOCAL NICHT IN SCHLEIFE
LOCAL NUR IN PROCEDURE
LOOP OHNE DO
MARKE DOPPELT DEFINIERT
NEXT OHNE FOR
PROCEDURE OHNE RETURN
PROCEDURE IN SCHLEIFE
PROCEDURE DOPPELT DEFINIERT
REPEAT OHNE UNTIL
RESUME IN FOR-NEXT-SCHLEIFE
RESUME OHNE PROCEDURE
RETURN OHNE PROCEDURE
SELECT OHNE ENDSELECT -> Nur in V3.0
SYNTAX FEHLER
WEND OHNE WHILE
WHILE OHNE WEND
ZEILE ZU LANG

BASIC-Fehlermeldungen

```
ERR:  TEXT (siehe ERR$):
 0    DIVISION DURCH NULL
 1    ÜBERLAUF
 2    ZAHL NICHT INTEGER|-2147483648 .. 2147483648
 3    ZAHL NICHT BYTE|0 .. 255
 4    ZAHL NICHT WORT|0 .. 65535                        -> V2.xx
      ZAHL NICHT WORT|-32768 .. 32767                   -> V3.0
 5    QUADRATWURZEL NUR FÜR|POSITIVE ZAHLEN
 6    LOGARITHMEN NUR FÜR|ZAHLEN > 0
 7    UNBEKANNTER FEHLER
 8    SPEICHER VOLL
 9    FUNKTION ODER BEFEHL|NOCH NICHT MÖGLICH
10    STRING ZU LANG. MAX. 32767 ZEICHEN
11    KEIN GFA-BASIC Vx.xx PROGRAMM
12    PROGRAMM ZU LANG|SPEICHER VOLL|NEW
13    KEIN GFA-BASIC-PROGRAMM|EOF - NEW
14    FELD ZWEIMAL DIMENSIONIERT
```

Nr.	Meldung	Version
15	FELD NICHT DIMENSIONIERT	
16	FELDINDEX ZU GROß	
17	DIM ZU GROß	
18	FALSCHE ANZAHL INDIZES	
19	PROCEDURE NICHT GEFUNDEN	
20	LABEL NICHT GEFUNDEN	
21	BEI OPEN NUR ERLAUBT¦	
	"I"NPUT "O"UTPUT "R"ANDOM¦"A"PPEND "U"PDATE	
22	FILE SCHON GEÖFFNET	
23	FILE # FALSCH	
24	FILE NICHT GEÖFFNET	
25	FALSCHE EINGABE, KEINE ZAHL	
26	FILEENDE ERREICHT¦EOF	
27	ZUVIEL PUNKTE FÜR¦POLYLINE/POLYFILL¦MAXIMAL 128	
28	FELD MUß EINDIMENSIONAL SEIN	
29	ANZAHL PUNKTE GRÖßER ALS FELD	
30	MERGE - KEIN ASCII-FILE	
31	MERGE - ZEILE ZU LANG - ABBRUCH	
32	==> SYNTAX NICHT KORREKT¦PROGRAMMABBRUCH	
33	MARKE NICHT DEFINIERT	
34	ZU WENIG DATA	
35	DATA NICHT NUMERISCH	
36	SYNTAXFEHLER IN DATA¦"" PAARWEISE VERWENDEN	-> Nur V2.xx
37	DISKETTE VOLL	
38	BEFEHL IM DIREKTMODUS¦NICHT MÖGLICH	
39	PROGRAMMFEHLER¦ KEIN GOSUB MÖGLICH	
40	CLEAR NICHT MÖGLICH IN¦FOR-NEXT-SCHLEIFEN	
	ODER¦PROZEDUREN	
41	CONT NICHT MÖGLICH	
42	ZU WENIG PARAMETER	
43	AUSDRUCK ZU KOMPLEX	
44	FUNKTION NICHT DEFINIERT	
45	ZU VIELE PARAMETER	
46	PARAMETER FALSCH¦KEINE ZAHL	
47	PARAMETER FALSCH¦KEIN STRING	
48	OPEN "R" - SATZLÄNGE FALSCH	
49	ZU VIELE "R"-FILES (MAX. 10)	-> V2.xx
	ZU VIELE "R"-FILES (MAX. 31)	-> V3.0
50	KEIN "R"-FILE	
51	NUR EIN FIELD ZU EINEM OPEN "R" MÖGLICH	-> Nur V2.xx
52	FIELDS GRÖßER ALS SATZLÄNGE	
53	ZU VIELE FIELDS (MAX. 9)	
	- nur in V2.xx -	
54	GET/PUT FIELD-STRING¦LÄNGE FALSCH	
55	GET/PUT SATZNUMMER FALSCH	
60	SPRITE-STRING-LÄNGE FALSCH	
61	FEHLER BEI RESERVE	
62	MENU FALSCH	
63	RESERVE FALSCH	
64	POINTER FALSCH	
65	FELDGRÖSSE < 256	-> Nur in V3.0
66	KEIN VAR-ARRAY	-> Nur in V3.0
67	ASIN/ACOS FALSCH	-> Nur in V3.0
68	FALSCHE VAR-TYPE	-> Nur in V3.0
69	ENDFUNC OHNE RETURN	-> Nur in V3.0
71	INDEX ZU GROSS	-> Nur in V3.0
90	FEHLER BEI LOCAL	
91	FEHLER BEI FOR	
92	RESUME (NEXT) NICHT MÖGLICH ¦FATAL, FOR ODER LOCAL	

```
 93     STAPEL-FEHLER                               -> Nur in V3.0
100     GFA-BASIC VERSION x.xx¦
          COPYRIGHT 1986-1988¦
        GFA SYSTEMTECHNIK GMBH
```

TOS-Fehlermeldungen

```
ERR:      TEXT (siehe ERR$):
 -1     * ALLGEMEINER FEHLER
 -2     * DRIVE NOT READY¦ZEITÜBERSCHREITUNG
 -3     * UNBEKANNTER BEFEHL
 -4     * CRC FEHLER¦DISK-PRÜFSUMME FALSCH
 -5     * BAD REQUEST¦UNGÜLTIGER BEFEHL
 -6     * SEEK ERROR¦SPUR NICHT GEFUNDEN
 -7     * UNKNOWN MEDIA¦BOOTSEKTOR FALSCH
 -8     * SEKTOR NICHT GEFUNDEN
 -9     * KEIN PAPIER
-10     * SCHREIBFEHLER
-11     * LESEFEHLER
-12     * ALLGEMEINER FEHLER 12
-13     * DISKETTE SCHREIBGESCHÜTZT
-14     * DISKETTE WURDE GEWECHSELT
-15     * UNBEKANNTES GERÄT
-16     * BAD SEKTOR (VERIFY)
-17     * ANDERE DISKETTE EINLEGEN
-32     * UNGÜLTIGE FUNKTIONSNUMMER
-33     * DATEI NICHT GEFUNDEN
-34     * PFADNAME NICHT GEFUNDEN
-35     * ZUVIELE DATEIEN OFFEN
-36     * ZUGRIFF NICHT MÖGLICH
-37     * UNGÜLTIGES HANDLE
-39     * SPEICHER VOLL
-40     * UNGÜLTIGE SPEICHERBLOCKADRESSE
-46     * UNGÜLTIGE LAUFWERKSBEZEICHNUNG
-49     * KEINE WEITEREN DATEN
-64     * GEMDOS BEREICHSFEHLER¦SEEK FALSCH?
-65     * INTERNER GEMDOS-FEHLER
-66     * KEIN BINÄRPROGRAMM
-67     * SPEICHERBLOCKFEHLER
```

Bomben-Fehlermeldungen

```
ERR :   TEXT (siehe ERR$):
102     2 BOMBEN - BUS ERROR¦PEEK/POKE FALSCH?
103     3 BOMBEN - ADRESS ERROR¦UNGERADE
          WORTADRESSE!¦DPOKE/DPEEK, LPOKE/LPEEK?
104     4 BOMBEN - ILLEGAL INSTRUCTION¦
          UNGÜLTIGER MASCHINENBEFEHL
105     5 BOMBEN - DIVIDE BY CERO¦68000
          DIVISION DURCH NULL
106     6 BOMBEN - CHK-EXEPTION¦68000
          CHK-BEFEHL
107     7 BOMBEN - TRAPV-EXEPTION¦68000
          TRAPV-BEFEHL
108     8 BOMBEN - PRIVILEGE VIOLATION¦68000
          PRIVILEGVERLETZUNG
```

```
109     9 BOMBEN - TRACE-EXEPTION|68000
          TRACE OHNE MONITOR
```

24.13 Syntax-Liste

Ein-/Ausgabebefehle

Dateneingabe

FORM INPUT Anz,Var$
FORM INPUT Anz AS Var$
Zeichen$=INKEY$
INPUT ["Text";,] Var1 [,Var2,...]
INPUT #Kanal,Var1 [,Var2,...]
A$=INPUT$(Anz)
A$=INPUT$(Anz,#Kanal)
LINE INPUT ["Text";,] Var$ [Var2$,...]
LINE INPUT #Kanal, Var$ [Var2$,...]

Datenausgabe

PRINT [AT(S,Z)][,']"Text"[[;,']Var[;,']Expr...;]
PRINT [#Kanal,[;]] "Text"[[;,']Var[;,']Expr...;]
PRINT USING "format",Expr [,Var,...][;]
PRINT USING Format$,Expr [,Var,...][;]
PRINT #Kanal,USING "format",Expr [,Var,...][;]
WRITE [#Kanal,] ["Text" [,Var,Expr;...]]

Bildschirmoperationen

HTAB Spalte
LOCATE S,Z
Var=POS(Dummy)
PRINT SPC(Anz)
PRINT [Ausdrücke;Werte;etc.;] SPC(Anz) [;etc.]
TAB(Position)
PRINT [Ausdrücke;Werte;etc.;] TAB(Anz) [;etc.]
VTAB Zeile

Diskettenoperationen

BLOAD "Dateiname" [,Start]
BSAVE "Dateiname",Start,Anz
CHAIN "Programmname"
CHDIR "Ordner"
CHDRIVE Laufwerk
CHDRIVE Pfad$
Var=DFREE(Station)
DIR ["Pfad"] [TO "Datei"]
Var$=DIR$(Laufwerk)
Var=EXIST(Dateiname)
Var=FGETDTA()

FILES ["Pfad"] [TO "Datei"]
Var=FSETDTA(Adresse)
Var=FSFIRST(Pfad$,Attribute)
Var=FSNEXT()
KILL "Dateiname"
LIST ["Dateiname"]
LOAD "Programmname"
MKDIR "Ordner"
NAME "Name_alt" AS "Name_neu"
PSAVE "Programmname"
RENAME "Name_alt" AS "Name_neu"
RMDIR "Ordner"
SAVE "Programmname"

Dateihandhabung

BGET [#]Kanal,Start,Anz
BPUT [#]Kanal,Start,Anz
CLOSE [#Kanal]
Var=EOF(#Kanal)
FIELD #Kanal,Anz AS Var1$ [,Anz AS Var2$,...]
FIELD #Kanal,Anz AT(Adr1) [,Anz AS Var$,...]
GET #Kanal [,Satznummer]
Var=LOC(#Kanal)
Var=LOF(#Kanal)
OPEN "Modus",#Kanal,"Dateiname" [,Satzlänge]
PUT #Kanal [,Satznummer]
RECALL #Kanal,Feld$(),Anz,Zeilenvar
RECORD [#] Kanal,Satznummer
RELSEEK #Kanal,[-] Offset
SEEK #Kanal,[-] Offset
TOUCH #Kanal
STORE #Kanal,Feld$() [,Anz]

Port-Ein-/-Ausgabe-Befehle

Var=INP?(Port)
Var=INP(Port)
Var=INP(#Kanal)
Var$=INPAUX$
Var$=INPMID$
Var=OUT?(Port)
OUT #Kanal,Bytewert
OUT Port,Bytewert
OUT #Kanal,Byte1 [,Byte2 [,Byte3,...]]
OUT Port,Byte1 [,Byte2 [,Byte3,...]]

Druckeranweisungen

HARDCOPY
LLIST
Var=LPOS(Dummy)
LPRINT [,'] "Text" [[;,'] Var1 [;,'] Expr...]

Sound-Chip-Anweisungen

SOUND Kanal,Volume,Note,Oktave,Dauer
SOUND Kanal,Volume,#Periode,Dauer
WAVE Kanal,Hüll,Form,Länge,Dauer

Programm-Struktur

Schleifenkonstruktionen

```
DO
... auszuführende Programmteile
LOOP

DO [WHILE Bedingung] [UNTIL Bedingung]
... auszuführende Programmteile, wenn DO-Bedingung
... wahr ist, bzw. solange LOOP-Bedingung
 wahr ist.
LOOP [WHILE Bedingung] [UNTIL Bedingung]

FOR Zaehl=Start TO [DOWNTO] Ende [STEP Schritt]
... auszuführende Programmteile
 NEXT Zaehl

REPEAT
... auszuführende Programmteile
UNTIL Bedingung

WHILE Bedingung
... auszuführende Programmteile...
WEND
```

Bedingte Verzweigungen

```
EXIT IF Bedingung

IF Bedingung [THEN]
... auszuführende Programmteile,
    wenn Bedingung    wahr ist
[ELSE
... auszuführende Programmteile,
    wenn Bedingung    unwahr ist ]
ENDIF

IF Bedingung1 [THEN]
... auszuführende Programmteile,
    wenn Bedingung1    wahr ist
[ELSE IF Bedingung2
... auszuführende Programmteile, wenn Bedingung1
    unwahr und Bedingung2    wahr ist.]
[ELSE IF Bedingung3
... auszuführende Programmteile, wenn alle
    vorherigen Bedingungen unwahr waren,
    jedoch Bedingung3    wahr ist.]
...
... ggfs. weitere ELSE IF-Abfragen
```

```
[ELSE
... auszuführende Programmteile, wenn alle
    vorherigen Bedingungen unwahr waren.]
ENDIF

SELECT Expr
CASE Konstante1 [TO Konstante2 [,[...] TO [...]]]
  ... auszuführende Programmteile, wenn Expr gleich
      Konstante1, bzw. - bei Option TO - wenn Expr
      innerhalb des Bereichs von Konstante1 bis
      Konstante2 liegt.
  [CONT]
[CASE Konstante1 [,Konstante2 [,Konstante3 [,...]]]
  ... auszuführende Programmteile, wenn 'Expr'     gleich
      Konstante1 oder gleich Konstante2 oder
      gleich Konstante3 oder ... oder ...]
  ... ggfs. weitere CASE-Entscheidungen
  [CONT]
[DEFAULT
  ... auszuführende Programmteile, wenn keine der
      vorhergehenden Abfragen zugetroffen hat.]
ENDSELECT
```

Bereichsdeklarationen

```
Befehlszeile ! Kommentartext
DATA [Wertedatas [,["] Textdatas ["],...]]
READ Var [,Var2,Var$,Var2$,...]
REM [Kommentar]
RESTORE [Labelname]
```

Variablendeklarationen

```
DEFBIT Defstring$
DEFBYT Defstring$
DEFFLT Defstring$
DEFINT Defstring$
DEFSTR Defstring$
DEFWRD Defstring$
```

Unterprogramme

```
DEFFN Funkt.name [(Var-Liste)]=Funkt.expr
FN Funktionsname [(Parameterliste)]

FUNCTION Name [(Var1,Var2%,Var3$,...[,VAR Var,...])]
  ... auszuführende Programmteile
  RETURN Back
ENDFUNC
GOSUB Prozedur [(Parameterliste)]
GOTO Label
LOCAL Loc.var [,Lokale Variablenliste,...]
ON Wert GOSUB Proc1 [,Proc2,Proc3,...]
ON Wert Proc1 [,Proc2,Proc3,...]
ON BREAK GOSUB Prozedur
ON BREAK
ON BREAK CONT
```

PROCEDURE Name [(Variablenliste)]
... auszuführende Programmteile
RETURN
PROCEDURE Name([Var1,...] VAR Varname1 [,Varname2$,...])
FUNCTION Name([Var1,...] VAR Varname1 [,Varname2$,...])

Assembler-/C-/PRG-Programmaufrufe

Var=C:Adressvar [(Parameterliste)]
CALL Adressvar [(Parameterliste)]
EXEC Modus,"Prg","Kom","Env"
Var=EXEC(Modus,"Prg","Kom","Env")
MONITOR [(Parameter)]
RCALL Adresse,Feld%()

Textoperationen

String-Manipulationen

MID$(Ziel$,Start [,Anz])="Text"
LSET Ziel$="Text"
RSET Ziel$="Text"

String-Analyse

Var=INSTR(Ziel$,Such$ [,Start])
Var=INSTR([Start,] Ziel$,Such$)
Var$=LEFT$(Ziel$ [,Anz])
Var=LEN(Var$)
Var$=MID$(Ziel$,Start [,Anz])
Var$=PRED(Expr$)
Var$=RIGHT$(Ziel$ [,Anz])
Var=RINSTR(Ziel$,Such$ [,Start])
Var=RINSTR([Start,] Ziel$,Such$)
Var$=SUCC(Expr$)

String-Formatierung

Var$=SPACE$(Anz)
Var$=STRING$(Anz,"Text")
Var$=STRING$(Anz,Ascii)
Var$=TRIM$(Ziel$)
Var$=UPPER$(Ziel$)

Arithmetikbefehle

Mathematische Operationen

ADD Var,Wert
DEC Var
DIV Var,Wert
INC Var

MUL Var,Wert
SUB Var,Wert
Var=ADD(Wert1,Wert2)
Var=DIV(Wert1,Wert2)
Var=MOD(Wert1,Wert2)
Var=MUL(Wert1,Wert2)
Var=SUB(Wert1,Wert2)

Numerische Funktionen

Var=ABS(Arg)
Var=EVEN(Arg)
Var=EXP(Arg)
Var=FIX(Arg)
Var=FRAC(Arg)
Var=INT(Arg)
Var=LOG[10](Arg)
Var=ODD(Arg)
Var=PRED(Arg)
Var=ROUND(Arg [,Stelle])
Var=SGN(Arg)
Var=SQR(Arg)
Var=SUCC(Arg)
Var=TRUNC(Arg)

Trigonometrische Funktionen

Var=ACOS(Arg)
Var=ASIN(Arg)
Var=ATN(Arg)
Var=COS(Arg)
Var=COSQ(Grad)
Var=DEG(Radian)
PI
Var=RAD(Grad)
Var=SIN(Arg)
Var=SINQ(Grad)
Var=TAN(Arg)

Vergleichsoperationen

Var=MAX(Expr1,Expr2 [,Expr3,...])
Var$=MAX(Expr1$,Expr2$ [,Expr3$,...])
Var=MIN(Expr1,Expr2 [,Expr3,...])
Var$=MIN(Expr1$,Expr2$ [,Expr3$,...])

Bit-Operationen

Var=AND(Wert1,Wert2)
Var=BCHG(Wert,Bit)
Var=BCLR(Wert,Bit)
Var=BSET(Wert,Bit)
Var=BTST(Wert,Bit)
Var=BYTE(Wert)
Var=CARD(Wert)
Var=EQV(Wert1,Wert2)

Var=IMP(Wert1,Wert2)
Var=OR(Wert1,Wert2)
Var=SHL(Wert,Bits)
Var=SHL&(Wert,Bits)
Var=SHL|(Wert,Bits)
Var=SHR(Wert,Bits)
Var=SHR&(Wert,Bits)
Var=SHR|(Wert,Bits)
Var=ROL(Wert,Bits)
Var=ROL&(Wert,Bits)
Var=ROL|(Wert,Bits)
Var=ROR(Wert,Bits)
Var=ROR&(Wert,Bits)
Var=ROR|(Wert,Bits)
Var=SWAP(Wert)
Var=WORD(Wert)
Var=XOR(Wert1,Wert2)

Zufallswerterzeugung

Var=RAND(n)
Var=RANDOM(n)
RANDOMIZE [(Start)]
Var=RND [(Arg)]

Grafik

Grafikdefinitionen

BOUNDARY Flag
COLOR Farbe
DEFFILL [Farbe],[Stil],[Muster]
DEFFILL [Farbe],Muster$
DEFLINE [Stil],[Dicke],[Form.a],[Form.e]
DEFMARK [Farbe],[Typ],[Größe]
DEFMOUSE Form
DEFMOUSE Maus$
DEFTEXT [Farbe],[Art],[Winkel],[Größe],[Face]
GRAPHMODE Modus
SETCOLOR Reg,Rot,Grün,Blau
SETCOLOR Reg,Mischwert
VSETCOLOR Reg,Rot,Grün,Blau
VSETCOLOR Reg,Mischwert

Objektgrafikbefehle

[P]BOX X_links,Y_oben,X_rechts,Y_unten
[P]CIRCLE X_cent,Y_cent,Radius [,Alpha,Beta]
[P]ELLIPSE Xcent,Ycent,Xrad,Yrad [,Alpha,Beta]
FILL Xpos,Ypos
FILL Xpos,Ypos [,Farbe]
POLYFILL Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]
POLYMARK Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]
[P]RBOX X_links,Y_oben,X_rechts,Y_unten
TEXT Xt,Yt [,[-]Länge],l"Text"

Strich-/Punktgrafik

DRAW TO Xpos,Ypos
DRAW X1,Y1 [TO X2,Y2 [TO X3,Y3...]]
DRAW Def$[,Const[,"Def"[,Var[,...]]]]
Var=DRAW(Index)
LINE X1,Y1,X2,X2
PLOT Xpos,Ypos
Var=POINT(Xpos,Ypos)
POLYLINE Pkte,Xp(),Yp() [OFFSET Xdiff,Ydiff]
SETDRAW Xpos,Ypos,Grad

Line-A-Grafikbefehle

ACHAR Ascii,Xpos,Ypos,Font,Art,Winkel
ACLIP Flag,X_li,Y_ob,X_re,Y_un
ALINE X1,Y1,X2,Y2,Farbe,Maske,Modus
APOLY P_adr,P_anz,Ymin TO Ymax,Farbe,Modus,M_adr,M_anz
ARECT X1,Y1,X2,Y2,Farbe,Modus,M_adr,M_anz
ATEXT Xpos,Ypos,Font,Text$
HLINE X1,Y,X2,Farbe,Modus,M_adr,M_anz
PSET Xpos,Ypos,Farbe
Var=PTST(Xpos,Ypos)

Grafikoperationen

CLIP Xpos,Ypos,Breite,Höhe [OFFSET X,Y]
CLIP Xl,Yo TO Xr,Yu [OFFSET X,Y]
CLIP #Windownummer [OFFSET X,Y]
CLIP OFFSET X,Y
CLIP OFF
GET X_links,Y_oben,X_rechts,Y_unten,Var$
HIDEM
PUT X_links,Y_oben,Var$ [,Modus]
RC_COPY Quell,Xq,Yq,Breite,Höhe TO Ziel,Xz,Yz [Modus]
Var=RC_INTERSECT(Xp1,Yp1,Br1,Hö1,Xp2,Yp2,Br2,Hö2)
SGET Var$
SHOWM
SPRITE Def.var$ [,Xpos,Ypos]
SPUT Var$
VSYNC

Datenumwandlung

Var=ASC("Zeichen")
Var$=BIN$(Expr)
Var$=BIN$(Expr [,Stellen])
Var=CFLOAT(Wert)
Var$=CHR$(Wert)
Var=CINT(Wert)
Var=CVI("2 Zeichen")
Var=CVL("4 Zeichen")
Var=CVS("4 Zeichen")
Var=CVF("6 Zeichen")
Var=CVD("8 Zeichen")
Var$=HEX$(Expr)
Var$=HEX$(Expr [,Stellen])

Var$=MKI$(16-Bit-Integer-Wert)
Var$=MKL$(32-Bit-Integer-Wert)
Var$=MKS$(Atari-BASIC-Realwert)
Var$=MKF$(V2.xx-GFA-BASIC-Realwert)
Var$=MKD$(V3.0-GFA-BASIC-Realwert)
Var$=OCT$(Expr)
Var$=OCT$(Expr [,Stellen])
Var$=STR$(Wert)
Var$=STR$(Wert [,Stellen [,Real]])
Var=VAL(Var$)
Var=VAL?(Var$)

Feld-, Speicher- und Zeigeroperationen

Feldoperationen

ARRAYFILL Feld(),Var
DELETE Feld(Index)
DELETE Feld$(Index)
DIM Arr1(Ind1 [,Ind2,...]) [,Arr2(Ind1 [,Ind2,...])...]
Var=DIM?(Feld())
ERASE Feld()
ERASE Feld1() [,Feld2() [,...]]
INSERT Feld(Index)=Wert
INSERT Feld$(Index)=Text
OPTION BASE Basis
QSORT Feld([Sign]) [,Anz [,Feld2%()]]
QSORT Feld$([Sign]) [WITH Vorgabe()] [,Anz [,Feld2%()]]
SSORT Feld([Sign]) [,Anz [,Feld2%()]]
SSORT Feld$([Sign]) [WITH Vorgabe()] [,Anz [,Feld2%()]]

Speicheroperationen

ABSOLUTE Var,Adresse
BITBLT Q_raster%(),Z_raster%(),R_def%()
BITBLT Blockadresse%
BITBLT Parameterfeld%()
BMOVE Quelle,Ziel,Anz
BYTE{Adresse}=Wert
CARD{gerade Adresse}=Wert
LONG{gerade Adresse}=Wert
Var=BYTE{Adresse}
Var=CARD{gerade Adresse}
Var=LONG{gerade Adresse}
CHAR{Adresse}=Expr$
Var$=CHAR{Adresse}
Realvar=DOUBLE{gerade Adresse}
Realvar=SINGLE{gerade Adresse}
DOUBLE{gerade Adresse}=Wert
SINGLE{gerade Adresse}=Wert
Realvar=FLOAT{gerade Adresse}
FLOAT{gerade Adresse}=Wert
Intvar=INT{Adresse}
Intvar=WORD{Adresse}
INT{Adresse}=Wert
WORD{Adresse}=Wert

Var=PEEK(Adresse)
Var=DPEEK(gerade Adresse)
Var=LPEEK(gerade Adresse)
POKE Adresse,Byte
DPOKE gerade Adresse,Word
LPOKE gerade Adresse,Long
SPOKE Adresse,Byte
SDPOKE gerade Adresse,Word
SLPOKE gerade Adresse,Long

Speicherverwaltung

INLINE Adresse%,Bytes
Back=MALLOC(Anz)
Back=MFREE(Adresse)
Back=MSHRINK(Adresse,Anz)
RESERVE Anz
RESERVE [[-]Anz]

Zeigeroperationen

Var=*Var
Var=*Feld()
Var=ARRPTR(Var$)
Var=ARRPTR(Feld())
Var=VARPTR(Var)

Programmkontrolle

Programmstart und -ende

CONT
EDIT
END
QUIT
QUIT [x]
RUN
RUN "Programmname"
STOP
SYSTEM
SYSTEM [x]

Löschfunktionen

CLEAR
CLR Var [,Var%,Var$,...]
CLS [#Kanal]
NEW

Zeitoperationen

Var$=DATE$
DATE$="Datum-String"
DELAY Sekunden

PAUSE Dauer
SETTIME Zeit$,Datum$
Var$=TIME$
TIME$="Zeit-String"
Var=TIMER

Fehlerbehandlung

Var=ERR
Var$=ERR$(Index)
ERROR Fehlernummer
Var=FATAL
ON ERROR GOSUB Prozedur
ON ERROR
RESUME
RESUME NEXT
RESUME Label

Auskünfte

Var=BASEPAGE
Var=CRSCOL
Var=CRSLIN
Var=FRE(Dummy)
Var=FRE()
Var=HIMEM
Var=L-A
Var=TYPE(Pointer)
Var=WORK_OUT(Index)

Tastaturkontrolle

KEYDEF Taste,Text
KEYGET Var
KEYLOOK Var
KEYPAD Vektor
KEYPRESS Code
KEYTEST Var

Multitasking

AFTER Ticks [GOSUB] Prozedur
AFTER CONT
AFTER STOP
EVERY Ticks [GOSUB] Prozedur
EVERY CONT
EVERY STOP

Debugging

DUMP [Defstring$] [TO Datei$]
Var$=TRACE$
TROFF
TRON [#Kanal]
TRON Prozedurname

Diverses

$ Text
DEFLIST Format
DEFNUM Stelle
Var=FALSE
LET Var=Wert
LET Var$=Text
MODE Modus
OPTION "Anweisung"
Var=TRUE
SWAP Var1,Var2
SWAP Element(x),Element(y)
SWAP Feld1(),Feld2()
SWAP *Pointer,Feld()
VOID Funktion

Interaktionen (Programm/Benutzer)

ALERT,Icon%,B_oxtext$,Def_Button%,Buttontext$,Backvar%
FILESELECT "Pfad","Auswahl",Backvar$
MOUSE Xpos,Ypos,Tasten
Var=MOUSEX
Var=MOUSEY
Var=MOUSEK
SETMOUSE Xpos,Ypos [,Button]
STICK(Modus)
Var=STICK(Maus-Port)
Var=STRIG(Maus-Port)

Window-Programmierung mit BASIC-Befehlen

CLEARW Nummer
CLEARW [#]Nummer
CLOSEW Nummer
CLOSEW [#]Nummer
FULLW Nummer
INFOW Nummer,"Text"
INFOW [#]Nummer,"Text"
OPENW Nummer [,Xcenter,Ycenter]
OPENW #Nummer,Xpos,Ypos,Breite,Höhe,Attribute
TITLEW Nummer,"Text"
TITLEW [#]Nummer,"Text"
TOPW #Nummer
Var=W_HAND(Nummer)
Var=W_INDEX(Handle)
Var=DPEEK(WINDTAB)
DPOKE WINDTAB,Wert
Var=WINDTAB(Nummer,Index)
WINDTAB(Nummer,Index)=Wert

Menüprogrammierung mit BASIC-Befehlen

MENU Menüpunkt,Attribut
MENU Array$()
MENU KILL
MENU OFF

Ereignisüberwachung mit BASIC-Befehlen

```
Var=MENU(Index)
ON MENU
ON MENU [Zeit]
ON MENU GOSUB Prozedurname
ON MENU BUTTON Anz,Taste,Status GOSUB Prozedurname
ON MENU IBOX Id,X,Y,B,H GOSUB Prozedurname
ON MENU OBOX Id,X,Y,B,H GOSUB Prozedurname
ON MENU KEY GOSUB Prozedurname
ON MENU MESSAGE GOSUB Prozedurname
```

GDOS/VDI-Bibliothek

```
Var=GDOS?
Var=V~H
V~H=Handle
Var=V_CLRVWK()
Var=V_CLSVWK()
Var=V_CLSWK()
Var=V_OPNVWK(Treiber_Id [,Linientyp,Linienfarbe,...
        ...Markertyp,Markerfarbe,Textstil,...
        ...Textfarbe,Füllstil,Füllmuster,...
        ...Füllfarbe,Koordinatenflag] )
Var=V_OPNWK(Treiber_Id [,Linientyp,Linienfarbe,...
        ...Markertyp,Markerfarbe,Textstil,...
        ...Textfarbe,Füllstil,Füllmuster,...
        ...Füllfarbe,Koordinatenflag] )
Var=V_UPDWK()
Var=VQT_EXTENT(Text$ [,Xlo,Ylo,Xro,Yro,Xru,Yru,Xlu,Ylu] )
Face=VQT_NAME(Index,Name$)
Var=VST_LOAD_FONTS(0)
Var=VST_UNLOAD_FONTS(0)
```

AES-Bibliothek

Application-Manager

```
Back=APPL_EXIT()
Back=APPL_FIND(Programmname)
Back=APPL_INIT()
Back=APPL_READ(Handle,Länge,Adresse)
Back=APPL_TPLAY(Adresse,Anzahl,Tempo)
Back=APPL_TRECORD(Adresse,Anzahl)
Back=APPL_WRITE(Handle,Länge,Adresse)
```

Event-Manager

```
Back=EVNT_BUTTON(Klicks,Taste,Status...
        ...[,<<Mausx,<<Mausy,<<Mausk,<<Switch])
Back=EVNT_DCLICK(Dklick,Modus)
Back=EVNT_KEYBD()
Back=EVNT_MESAG(Adresse)
Back=EVNT_MOUSE(Flag,Xpos,Ypos,Breite,Höhe...
    ...[,<<Mausx,<<Mausy,<<Mausk,<<Switch])
```

```
Back=EVNT_MULTI(Modus,Klicks,Taste,Status,Bflag1,...
        ...Xpos1,Ypos1,Breite1,Höhe1,Bflag2,...
        ...Xpos2,Ypos2,Breite2,Höhe2,Adresse,...
          ...Zeit [,<<Mausx,<<Mausy,<<Mausk,...
                 ...<<Switch,<<Bklicks])
Back=EVNT_TIMER(Zeit)
```

Formular-Manager

```
Back=FORM_ALERT(Button,Text$)
Back=FORM_BUTTON(Adresse,Objekt,Klicks,<<N_objc)
Back=FORM_CENTER(Adresse,<<Xpos,<<Ypos,<<Breite,<<Höhe)
Back=FORM_DIAL(Modus,Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)
Back=FORM_DO(Adresse,Objekt1)
Back=FORM_ERROR(Code)
Back=FORM_KEYBD(Adresse,Objekt1,Objekt2,Char$,...
                 ...<<N_objc,<<N_char$)
```

Fileselect-Manager

```
Back=FSEL_INPUT(<<Pfad$>>,<<Datei$>>,<<Button)
GRAPHICS-MANAGER
Back=GRAF_DRAGBOX(Litb,Lith,Xsta,Ysta,Xpos,Ypos,...
               ...Breite,Höhe,<<Lastx,<<Lasty)
Back=GRAF_GROWBOX(Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)
Back=GRAF_HANDLE(<<Ch_breite,<<Ch_höhe,<<Bx_breite,<<Bx_höhe)
Back=GRAF_MKSTATE(<<Mausx,<<Mausy,<<Mausk,<<Switch)
Back=GRAF_MOUSE(Maus,Adresse)
Back=GRAF_MOVEBOX(Breite,Höhe,Xsta,Ysta,Xziel,Yziel)
Back=GRAF_RUBBERBOX(Xpos,Ypos,Bmin,Hmin,<<Lastb,<<Lasth)
Back=GRAF_SHRINKBOX(Litx,Lity,Litb,Lith,Bigx,Bigy,Bigb,Bigh)
Back=GRAF_SLIDEBOX(Adresse,Rahmen,Slider,Flag)
Back=GRAF_WATCHBOX(Adresse,Objekt,Stat_i,Stat_o)
```

Menü-Manager

```
Back=MENU_BAR(Adresse,Modus)
Back=MENU_ICHECK(Adresse,Objekt,Modus)
Back=MENU_IENABLE(Adresse,Objekt,Modus)
Back=MENU_TEXT(Adresse,Objekt,Text$)
Back=MENU_TNORMAL(Adresse,Objekt,Modus)
Back=MENU_REGISTER(Handle,Text$)
```

Objekt-Manager

```
Back=OBJC_ADD(Adresse,Elter,Kind)
Back=OBJC_CHANGE(Adresse,Objekt,Dummy,Xpos,Ypos,...
                ...Breite,Höhe,Status,Modus)
Back=OBJC_DELETE(Adresse,Objekt)
Back=OBJC_DRAW(Adresse,Ebene1,Kinder,Xpos,Ypos,Breite,Höhe)
Back=OBJC_EDIT(Adresse,Objekt,Char$,Zpos,Modus,<<N_pos)
Back=OBJC_FIND(Adresse,Objekt1,Ebenen,Xpos,Ypos)
Back=OBJC_OFFSET(Adresse,Objekt,<<Xpos,<<Ypos)
Back=OBJC_ORDER(Adresse,Objekt,Ebene)
```

Resource-Manager

```
Back=RSRC_FREE()
Back=RSRC_GADDR(Typ,Objekt,<<Adresse)
Back=RSRC_LOAD(Rsc_name$)
Back=RSRC_OBFIX(Adresse,Objekt)
Back=RSRC_SADDR(Typ,Objekt,Zeiger)
```

Scrap-Manager

```
Back=SCRP_READ(<<Puffer$>>)
Back=SCRP_WRITE(Text$)
```

Shell-Manager

```
Back=SHEL_ENVRN(<<Adresse,String$)
Back=SHEL_FIND(<<Pfad$>>)
Back=SHEL_GET(Anzahl,<<Puffer$)
Back=SHEL_PUT(Anzahl,Envrn$)
Back=SHEL_READ(<<Name$,<<Comm$)
Back=SHEL_WRITE(Modus,Grafik,Gemflag,Comm$,Name$)
```

Window-Manager

```
Back=WIND_CALC(Modus,Attribut,Inx,Iny,Inb,Inh,...
          ...<<Outx,<<Outy,<<Outb,<<Outh)
Back=WIND_CLOSE(Handle)
Back=WIND_CREATE(Attribut,Max_xl,Max_yo,Max_br,Max_ho)
Back=WIND_DELETE(Handle)
Back=WIND_FIND(Xpos,Ypos)
Back=WIND_GET(Handle,Modus,<<Get1,<<Get2,<<Get3,<<Get4)
Back=WIND_OPEN(Handle,Xpos,Ypos,Breite,Höhe)
Back=WIND_SET(Handle,Modus,Set1,Set2,Set3,Set4)
Back=WIND_UPDATE(Modus)
```

Objektvariablen

```
OB_ADR(Adresse,Objekt)=Wert
Wert=OB_ADR(Adresse,Objekt)
Wert=OB_NEXT(Adresse,Objekt)
OB_NEXT(Adresse,Objekt)=Wert
Wert=OB_HEAD(Adresse,Objekt)
OB_HEAD(Adresse,Objekt)=Wert
Wert=OB_TAIL(Adresse,Objekt)
OB_TAIL(Adresse,Objekt)=Wert
Wert=OB_TYPE(Adresse,Objekt)
OB_TYPE(Adresse,Objekt)=Wert
Wert=OB_FLAGS(Adresse,Objekt)
OB_FLAGS(Adresse,Objekt)=Wert
Wert=OB_STATE(Adresse,Objekt)
OB_STATE(Adresse,Objekt)=Wert
Wert=OB_SPEC(Adresse,Objekt)
OB_SPEC(Adresse,Objekt)=Wert
Wert=OB_X(Adresse,Objekt)
OB_X(Adresse,Objekt)=Wert
Wert=OB_Y(Adresse,Objekt)
```

OB_Y(Adresse,Objekt)=Wert
Wert=OB_W(Adresse,Objekt)
OB_W(Adresse,Objekt)=Wert
Wert=OB_H(Adresse,Objekt)
OB_H(Adresse,Objekt)=Wert

GEM-Adressen/-Felder

Var%=LPEEK(ADDRIN+Offset)
LPOKE ADDRIN+Offset,4-Byte-Adresse
Var%=ADDRIN(Index)
ADDRIN(Index)=4-Byte-Adresse
Var%=LPEEK(ADDROUT+Offset)
LPOKE ADDROUT+Offset,4-Byte-Adresse
Var%=ADDROUT(Index)
ADDROUT(Index)=4-Byte-Adresse
Var=DPEEK(CONTRL+Offset)
DPOKE CONTRL+Offset,2-Byte-Wert
Var&=CONTRL(Index)
CONTRL(Index)=2-Byte-Wert
Var%=LPEEK(GB+Offset)
LPOKE GB+Offset,4-Byte-Adresse
Var=DPEEK(GCONTRL+Offset)
DPOKE GCONTRL+Offset,2-Byte-Wert
Var&=GCONTRL(Index)
GCONTRL(Index)=2-Byte-Wert
Var=DPEEK(GINTIN+Offset)
DPOKE GINTIN+Offset,2-Byte-Wert
Var&=GINTIN(Index)
GINTIN(Index)=2-Byte-Wert
Var=DPEEK(GINTOUT+Offset)
DPOKE GINTOUT+Offset,2-Byte-Wert
Var&=GINTOUT(Index)
GINTOUT(Index)=2-Byte-Wert
Var=DPEEK(INTIN+Offset)
DPOKE INTIN+Offset,2-Byte-Wert
Var&=INTIN(Index)
INTIN(Index)=2-Byte-Wert
Var=DPEEK(INTOUT+Offset)
DPOKE INTOUT+Offset,2-Byte-Wert
Var&=INTOUT(Index)
INTOUT(Index)=2-Byte-Wert
Var=DPEEK(PTSIN+Offset)
DPOKE PTSIN+Offset,2-Byte-Wert
Var&=PTSIN(Index)
PTSIN(Index)=2-Byte-Wert
Var=DPEEK(PTSOUT+Offset)
DPOKE PTSOUT+Offset,2-Byte-Wert
Var&=PTSOUT(Index)
PTSOUT(Index)=2-Byte-Wert
Var=VDIBASE

Systemaufrufe

Var=BIOS(Opcode [,Parameterliste])
Var=GEMDOS(Opcode [,Parameterliste])
GEMSYS [[(] Opcode [)]]

VDISYS [[(] Opcode [)]]
VDISYS [[(] Opcode [,I_len,P_len[,Id]][)]]
Var=XBIOS(Opcode [,Parameterliste])

24.14 Alphabetische Befehlsliste

Typ:

(a)	Adresse
(b)	Befehl
(d)	Declaration
(e)	Einstellung
(f)	Funktion
(o)	Operator
(v)	Variable

Index

Der Mega ST, das Flaggschiff von Atari, verfügt über eine Vielzahl von Anwendungsmöglichkeiten, die sich u. a. aus dem extrem großen Arbeitsspeicher dieses Rechners ergeben. Insbesondere in den Bereichen Desktop Publishing (DTP) und Programmierung erweist sich der Mega ST als vielseitig und anderen Geräten weit überlegen. In diesem Buch gehen die Autoren ausführlich auf die Möglichkeiten dieses Rechners ein und weisen darüber hinaus auf alle technischen Einzelheiten hin, deren Kenntnis für jeden Mega-ST-Besitzer unerläßlich ist. Besonders interessant ist es für alle, die das dokumentierte Blitter-TOS für die eigene Programmentwicklung nutzen wollen.



Aus dem Inhalt:

- Die Bedienung und die Handhabung des Mega ST
- Die Software für den Mega ST (Textverarbeitungen, Datenbanken, Tabellenkalkulationen usw.)
- Desktop Publishing mit dem Mega ST (Beckerpage ST usw.)
- Die Programmierung des Mega ST (AES, VDI, BIOS, XBIOS, GEMDOS usw.)
- Die Periphierie des Mega ST (Festplatte, Laserdrucker usw.)
- Der Aubau des Rechners
- Die Nutzung des großen RAM-Speichers - RAM-Disk bis 3,5 MByte mit Autokonfiguration
- Nützliche Beispielprogramme und ein ausführlich dokumentiertes Blitter-TOS-Listing

Dittrich, Englisch, Severin
Das große Mega ST Buch
Hardcover, 512 Seiten, DM 69,–
ISBN 3-89011-196-3

DAS STEHT DRIN:

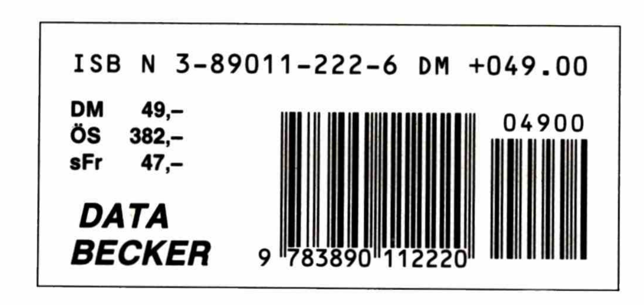
Endlich gibt es GFA V3.0! Zu diesem umfangreichen BASIC gehört auch ein umfangreiches Buch, in dem detailliert jeder Befehl behandelt wird. Dabei liefert es keine nackte Befehlsübersicht, sondern wirklich brauchbares Material in Hülle und Fülle. Anhand zahlreicher Beispielprogramme lernen Sie dieses leistungsfähige BASIC spielend zu beherrschen.

Aus dem Inhalt:

- Das Editor-Menü
- Variablentypen und -organisation
- Diskettenoperationen
- Strukturierte Programmierung
- Mausabfrage in eigenen Programmen
- Sound-Programmierung
- Beschreibung des Resource-Construction-Set
- Verwendung von Multitasking-Befehlen
- Programmieren von Pull-Down-Menüs
- Abfrage von Ereignissen (Events)
- Window-Programmierung
- Zugriff auf GEMDOS, BIOS und XBIOS
- Komplette AES- und VDI-Library-Beschreibung
- Verwenden eigener Fonts mit GDOS
- Eine komplette Adressenverwaltung als RAM-Kartei
- Komplette Befehlsübersicht über alle Befehle des GFA-BASIC

UND GESCHRIEBEN HABEN DIESES BUCH:

Uwe Litzkendorf, Student der Architektur, ist erfahrener Programmierer und arbeitet seit langem mit GFA-BASIC. Mit diesem Buch erscheint bereits sein dritter Titel zu diesem Thema.





Scan von TPAU